Standards Conformance Guide

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 802-1953-10
Revision A, November 1995

SunSoft
A Sun Microsystems, Inc. Business

Please
Recycle

Adobe PostScript

Contents

Preface

Solaris™ is Sun Microsystem's integrated computing environment that includes the SunOS™ operating system, OpenWindows™ and numerous bundled utilities. SunOS is compliant with the System V Interface Definition, Issue 3 from UNIX® System Laboratories.

This book is part of the Solaris documentation set. It discusses the compliance of Solaris 2.5 and the SunOS 5.5 operating system to the following specifications and standards:

- Application Binary Interface (ABI)
- System V Interface Definition, Issue 3 (SVID)
- Device Driver Interface / Driver-Kernel Interface (DDI / DKI)
- Data Link Provider Interface (DLPI)
- Transport Provider Interface (TPI)
- OPEN LOOK® Graphical User Interface (GUI)
- OSF / Motif 1.2
- X Window System™ Protocol, Version 11 (X11)
- X / Open™ XPG3 BASE
- X / Open XPG4 BASE
- X / Open UNIX® '93
- ANSI / IEEE Standard 1003.1c–1995 – (POSIX.1)
- ANSI / IEEE Standard 1003.2–1992 – (POSIX.2)
- ANSI C Programming Language
- International Standards Organization (ISO) 8859-1
- Federal Information Processing Standard 151-2
- Federal Information Processing Standard 158

- ANSI/IEEE Standard 754
- SPARC Compliance Definition 2.1 (SCD 2.1)

Chapter 1 of this book introduces the organizations responsible for the specifications and standards covered in this guide. Chapters 2 through 10 discuss the specifications and standards; a brief account of each specification or standard is followed by a statement of compliance with it.

A Look at Some Standardization Organizations 1

This chapter briefly discusses the histories of organizations responsible for the specifications and standards discussed in this guide. SunSoft recognizes the importance of compliance with existing and evolving standards and is firmly committed to support and participate in ongoing efforts toward standardization.

Institute of Electrical and Electronics Engineers (IEEE) and POSIX

A group of UNIX systems users, /usr/group®, established a committee with the objective of proposing a set of standards for application level interfaces. After publishing the 1984 /usr/group Standard, the group decided to seek international status for the standard. In early 1984, the /usr/group Standards Committee closed its activities in its own name and its members were encouraged to become involved in the IEEE POSIX committee so that the work could become the basis for an official international standard.

The first externally visible result of this initiative was the publication of the IEEE Trial-Use Standard in March 1986. Formal approval followed in August 1988 of IEEE Standard 1003.1-1988, a “Portable Operating System Interface for Computer Environments” (POSIX), which became the first step toward a truly portable operating system standard.

Although originally planned to refer to the IEEE Standard 1003.1-1988, the name POSIX has come to refer to the whole family of related standards and parts of the International Standard ISO/IEC 9945. POSIX.1 has emerged as the preferred reference to IEEE Standard 1003.1-1990. An update to the 1988 standard, IEEE Standard 1003.1-1990, was also adopted as International Standard ISO/IEC 9945-1:1990 by the International Organization for Standardization (ISO) and by the International Electrotechnical Commission (IEC). In 1993, POSIX.1 was amended to include extensions in support of real-time applications. Chapter 8 of this manual discusses the compliance of Solaris software with IEEE Standard 1003.1b-1993.

In 1992, IEEE Standard 1003.2–1992 became part of the POSIX series of standards. Referred to as "POSIX.2," IEEE Standard 1003.2–1992 defines a standard interface and environment for applications that require a shell command language interpreter and a set of common utility programs. Chapter 9 of this manual discusses the compliance of Solaris software with IEEE Standard 1003.2–1992.

Note – Use of an IEEE standard is voluntary.

X/Open

Founded in 1984, X/Open™ is a worldwide consortium of system vendors, ISVs, and users, organized to adopt existing standards and adapt them into a consistent environment called the Common Applications Environment (CAE). Through establishment of the CAE and awarding of the X/Open brand trademark to products that comply with the X/Open definitions, X/Open aims to ensure portability and connectivity of applications. Where there is no official standard, it is X/Open policy to work closely with standards bodies to encourage the emergence of common standards.

Many of the world's major hardware suppliers, including Sun Microsystems, are X/Open members. Most of X/Open's technical work is accomplished by personnel from its member companies.

X/Open publishes its specifications in the X/Open Portability Guide (XPG). XPG defines the interfaces identified as components of the Common Applications Environment. It contains an evolving portfolio of practical applications programming interfaces (APIs), which enhance portability of application programs at the source code level. The interfaces are supported by an extensive set of conformance tests and the distinct X/Open brand trademark. The X/Open Portability Guide, Issue 3 (XPG3) encompasses the IEEE POSIX.1 operating system interface and numerous extensions. Issue 4 (XPG4) encompasses the POSIX.2 shell and utilities.

In 1993, ownership of this UNIX trademark was transferred to X/Open Company, Ltd.

Chapter 5 of this guide discusses the compliance of Solaris to the programming interface specifications presented in the X/Open Portability Guide, Issue 3.

Chapter 6 of this guide discusses the compliance of Solaris to the programming interface specifications presented in the X/Open Portability Guide, Issue 4.

Chapter 7 of this guide discusses the compliance of Soilaris to the X/Open UNIX ‘93 Brand.

National Institute of Standards and Technology (NIST)

The National Institute of Standards and Technology, (formerly the National Bureau of Standards), is a federal government agency that issues Federal Information Processing Standards Publications (FIPS PUBS). Standards are first approved by the Secretary of Commerce according to Section 111(d) of the Federal Property and Administrative Services Act of 1949, as amended by the Computer Security Act of 1987, Public Law 100-235.

International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC)

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) together form a system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of international standards through technical committees established by ISO and IEC to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other governmental and nongovernmental international organizations also take part in the work.

The X Consortium

The X Consortium, Inc. is a non-profit organization that solicits, develops, standardizes, distributes, and maintains a collection of technologies that come under the umbrella term of the “X Window System.” The technologies are devoted to graphic presentation and human interaction that can be distributed over computer networks. The X Consortium is sustained by corporate membership fees. This organization has recently broadened its original charter by taking on a role as prime contractor to the Open Software Foundation in the development of licensable software in the form of Motif and the Common Desktop Environment.

UniForum

UniForum, formerly /usr/group, is an association of individuals, corporations and institutions with an interest in open systems. This organization provides input to POSIX and other standards committees and consortia to aid in the development of independent industry-driven standards. UniForum has more than 10,000 members representing a cross-section of the UNIX system community. The membership includes hardware manufacturers, vendors of operating systems and software development tools, software designers, consultants, academics, authors and applications programmers, among others.

American National Standards Institute (ANSI)

The American National Standards Institute (ANSI) verifies that requirements for due process, consensus, and other criteria for approval have been met by the standards developer before it grants a standard approval as an American National Standard.

Consensus is established when the ANSI Board of Standards Review determines that the criteria for standards approval has been met by the standards development organizations. Consensus requires that all views and objections be considered and that a concerted effort be made toward their resolution.

ANSI does not develop standards, nor does it interpret any American National Standards.

Note – Use of an American National Standard is voluntary.

The Open Software Foundation

The Open Software Foundation (OSF™) was founded in 1988 with the goal of bringing global acceptance to a single user interface standard for open software. The OSF Motif graphical user interface bases its foundation on industry standards such as POSIX and X/Open.

In its efforts to develop a single user interface standard, OSF uses an open process to solicit technologies from hardware and software vendors worldwide. The selection criteria used by OSF include technical excellence, maturity, compatibility with established industry standards, and the ability to perform in a heterogeneous networked environment that may incorporate a variety of platforms.

UNIX System V Release 4-Based (SVR4) Specifications 2

This chapter provides an introduction to UNIX System V Release 4, discusses related specifications, and identifies how the SunOS operating system conforms to those specifications.

UNIX System V Release 4

The UNIX operating system was developed by Ritchie and Thompson at Bell Laboratories in the early 1970s. From 1977 to 1982, Bell Laboratories combined several variants of the UNIX system devised by American Telephone and Telegraph (AT&T), into a single system, known commercially as UNIX System III. Bell Laboratories later added several features to UNIX System III, calling the new product UNIX System V, and AT&T announced official support for System V in January 1983.

UNIX System V Release 4 (SVR4), the result of a cooperative venture entered into by Sun Microsystems and AT&T, was announced in November of 1989. SVR4 is a synthesis of the best functionality of AT&T's UNIX System V Release 3, Berkeley Software Distribution 4.3, Sun's SunOS releases, and Microsoft's XENIX®. SVR4 provides the notions of a consistent Application Programming Interface (API) and a single Application Binary Interface (ABI) for each hardware platform. It offers scalability that allows users, depending on their needs, to move to larger or smaller machines while still using the same environment.

Application Binary Interface

The System V Application Binary Interface (ABI) defines a standard binary interface for compiled applications on systems that implement UNIX System V Release 4 or other operating systems that comply with the System V Interface Definition, Third Edition.[1]

The ABI defines a binary interface for application programs that are compiled and packaged for System V implementations on different hardware architectures. Because a binary specification must include information particular to the computer processor architecture for which it is intended, it is not possible for a single document to specify the interface for all possible System V implementations. Therefore, the System V ABI is a family of specifications.

The System V ABI is composed of two basic parts: a generic part that is a source-level interface which describes those aspects that remain constant across all hardware implementations of System V, and a processor-specific part that provides a complete binary interface for specific CPU architectures. Together, the generic ABI and the processor-specific supplement for a single hardware architecture provide a complete interface specification for compiled application programs on systems that share a common hardware architecture.

Software that is ABI-compliant for a particular architecture runs unchanged in its binary form on any ABI-compliant machine of that architecture. Also, this software is source compatible with any other ABI-compliant system and runs unchanged after compilation on the target system.

SunOS Compliance With the ABI Specification

The SunOS operating system is compliant with both the generic ABI as defined in the *AT&T System V Application Binary Interface: Generic ABI* (ISBN 0-13-100439-5) and the processor-specific part of the ABI as defined in the *Application Binary Interface SPARC Processor Supplement* (ISBN 0-13-104696-9) or the *Application Binary Interface Intel 386 Processor Supplement* (ISBN 0-13-104670-5), depending on the underlying hardware architecture.

1. The System V Interface Definition, Third Edition is also referred to as SVID89 and SVID3.

ABI Specifications and Related Publications

The following documents comprise the ABI specification for the SunOS operating system:

- *AT&T System V Application Binary Interface: Generic ABI*
- *AT&T System V Application Binary Interface SPARC Processor Supplement*
- *AT&T System V Application Binary Interface Intel 386 Processor Supplement*

The documents listed above refer to other specifications and standards, some of which are identified below:

- *The System V Interface Definition, Third Edition*
- *The IEEE Std. 1003.1-1990 Portable Operating System Interface (POSIX.1)-Part 1: System Application Program Interface [C Language]*
- *The IEEE Std 754-1985 Floating Point Processing Specification*
- *The X/Open Portability Guide, Issue 3*
- *The X/Open Portability Guide, Issue* 4
- *The ANSI Std. X3.159-1989 C Language Specification*
- *The X11 X Window System Graphical User Interface Specification*
- *The SPARC Architecture Manual, Version 8*
- *i486 MICROPROCESSOR Programmer's Reference Manual*
- *80386 Programmer's Reference Manual*
- *80387 Programmer's Reference Manual*

These specifications and standards are discussed elsewhere in this manual.

System V Interface Definition (SVID)

The System V Interface Definition (SVID), first published by AT&T in 1985, represented a major standards initiative. AT&T was a prominent member of /usr/group and the influence of /usr/group is evident in the SVID.

The SVID specifies an operating system environment that allows users to create applications software that is independent of any particular computer hardware. It specifies the operating system components available to both end-users and application programs and defines the functionality, but not the implementation, of components. The SVID specifies the source code interfaces of each operating system component, as well as the runtime behavior seen by an application program or an end-user.

An application using only components defined in the SVID will be compatible with and portable to any computer that supports the SVID. The SVID is organized into a Base System Definition with a series of Extension Definitions. The Base System Definition specifies the components that all System V operating systems must provide. The extensions to the Base System are not required.

All conforming systems must support the source-code interfaces and runtime behavior of all the components of the Base System. A system may conform to none or some extensions. All of the required components must be present for a system to meet the requirements of the extension.

SunOS Compliance With SVID3

The SunOS operating system is compatible with the Base System of the System V Interface Definition, Third Edition. Writing to SVID3 ensures that your applications will be source compatible. Applications that are SVID3 compliant will compile and run on the SunOS operating system.

The SunOS operating system meets all SVID requirements for the following: Base System, Basic Utilities, Kernel, Network Services, Terminal Interface Extensions, Advanced Utilities, and Software Development Extensions.

SVID Specification

System V Interface Definition, Third Edition, Volumes 1-4, AT&T

Device Driver Interface/Driver-Kernel Interface (DDI/DKI)

The Solaris 2.4 DDI, Device Driver Interface and DKI, Driver-Kernel Interface, comprise a set of standard interfaces for device drivers. SVR4 requires that each vendor provide and document a hardware-specific DDI. The Solaris 2.4 DDI is a set of device driver interfaces defined by SunSoft that meets that requirement. The DKI is intrinsic to System V, Release 4 (SVR4). The DKI is divided into two parts: the set of interfaces called *DDI/DKI* that will continue to be supported in future release of System V and the set of interfaces called *DKI only* that may not be supported in the future.

SunOS Compliance with the Specification

The SunOS implementation of the DDI / DKI and DKI-only interfaces for device drivers is compliant with the specification described in the *UNIX System V Release 4 Device Driver Interface/Driver-Kernel Interface (DDI/DKI) Reference Manual*.

DDI/DKI and DKI Specifications and Related Publications

The following manuals describe the DDI and DKI interfaces.

- DDI interfaces are described in the *Writing Device Drivers*.
- For detailed information on how to write device drivers to these interfaces, see *Writing Device Drivers*.
- The DKI interfaces are specified in *The UNIX System V Release 4 Device Driver Interface/Driver-Kernel Interface (DDI/DKI) Reference Manual*.

Data Link Provider Interface (DLPI)

The SVR4 STREAMS-based Data Link Provider Interface (DLPI) is a kernel-level interface that supports the services of the Data Link Layer for both connection-mode and connectionless-mode services. The specification, *A STREAMS-Based Data Link Provider Interface*, Version 2, designates the format for a set of messages between the data link provider and the data link user. The DLPI header, `<dlpi.h>`, is part of SVR4 and is included with this SunOS release.

DLPI enables a data link service user to access and use any of a variety of conforming data link service providers without special knowledge of the provider’s protocol. Specifically, the interface is intended to support X.25, LAPB, BX.25 level 2, SDLC, ISDN, LAPD, Ethernet™, CSMA / CD, token ring, token bus, Bisync, FDDI, and other data link protocols.

SunOS Compliance With DLPI

The SunOS operating system is compliant with the DLPI specification, Version 2, 1991, revised by the Open Systems Interconnection Working Group (OSIWG), a working group within UNIX International (UI). The version 2 `<dlpi.h>` header is delivered with SunOS.

DLPI Specification

The following specification is based on the DLPI specification and includes version 2 of the `<dlpi.h>` header.

A STREAMS-Based Data Link Provider Interface -Version 2, UNIX International

Transport Provider Interface

The Transport Provider Interface (TPI) consists of the kernel components of the Transport Level Interface. TPI specifies the transport service interface in terms of STREAMS messages. The TPI structure is described in the TPI specification for System V Release 4.0.

SunOS Compliance with TPI

The SunOS operating system is entirely compliant with the Transport Provider Interface and intends to remain compliant as TPI continues to evolve. The X/Open Transport Interface (XTI), which was based on and evolved from the Transport Level Interface (TLI), will influence the evolution of TPI. (It is the intention of SunSoft to be compliant with XTI in the near future.)

X11, PostScript and Sun Microsystems' OpenWindows 3

This chapter discusses the OPEN LOOK development environment, the X Window System, Version 11 (X11) and the parts of OpenWindows 3.5 that implement aspects of X11.

OPEN LOOK

The OPEN LOOK Graphical User Interface (GUI) was developed by Sun Microsystems in partnership with AT&T. In July 1988, Sun and AT&T distributed more than 1000 copies of the OPEN LOOK specification draft to UNIX system users for review. The comments received from the industry were used to create the final version of the OPEN LOOK specification.

OPEN LOOK is a specification for a user interface within a window environment based on the pioneering work done on graphical user interfaces at Xerox PARC in the 1970s. A graphical user interface standard describes how applications appear on the screen and their behavior in relation to the user. The OPEN LOOK GUI was designed to provide a simple, consistent, and efficient interface.

The OPEN LOOK GUI uses windows and menus with common graphic symbols instead of typed system commands to provide an intuitive environment with a consistent screen layout that can be used across various platforms and operating systems.

OPEN LOOK compliance has two components: toolkit compliance and environment compliance. There are three types of software that fit within these two categories: toolkits, applications, and environments. Because most applications are built using toolkits, they usually assume the level of compliance characteristic of the toolkit used to create them. Toolkits, applications and environments as understood in OPEN LOOK parlance are described below:

- Toolkits. A toolkit is a set of programming components used to build OPEN LOOK GUI applications. It consists of a high-level programming interface that provides the elements required to build a user interface. Toolkits provide a set of routines that implement the various interface elements as defined by the specification. The application developer uses the routines provided by the toolkit to create and position the interface elements as needed. The toolkit makes application development easier and ensures that the user interface remains consistent by using the same building blocks. Toolkits help developers create user-interface prototypes by providing a simple and easy-to-use programming interface.
- Applications. An application is a program or set of programs designed to perform a specific task. Applications are built using a user-interface toolkit or other developer tools. While it is possible for the application developer to implement the OPEN LOOK User Interface (UI) without a toolkit, the usual approach is to use a toolkit written for a specific windowing platform. Together, applications and their use of the toolkit define the way programs look and feel to users.
- Environments. An environment is a program or set of programs that effect the design and operation of an OPEN LOOK GUI implementation. An OPEN LOOK compliant environment consists of an OPEN LOOK User Interface (UI) window manager, file manager, workspace properties window, and other utility programs.

To guarantee an OPEN LOOK GUI-compliant application, the developer must write the application with an OPEN LOOK GUI compliant toolkit and run the application in a compliant OPEN LOOK GUI environment.

Trademark licensing is available for the following three levels of OPEN LOOK certification:

- Level 1: This level contains all the essential components of a complete user interface. It delineates the minimum features required to certify an implementation as OPEN LOOK GUI compliant. Among the features covered in Level 1 are window types and properties, menu formats, and mouse and keyboard.
- Level 2: Compliance with Level 2 requires the presence of all of the features comprising Level 1 and additional features mandatory for Level 2. These include abbreviated buttons, nonstandard window types, scrollbars, and icon settings for color implementations.
- Level 3: Level 3 is a superset of Level 2. This level requires that an application contain certain specialized features and a process manager for extending the functionality of the OPEN LOOK GUI.

For a complete list of the required features for each level, see “Appendix A, Certification” in the *OPEN LOOK Graphical User Interface Functional Specification*.

Solaris Compliance With OPEN LOOK

The Sun GUI is a superset of OPEN LOOK and includes Sun value-added features. In moving toward full support of the Sun GUI, all required OPEN LOOK features will be supported by Sun at all levels.

OpenWindows implements the OPEN LOOK GUI standard. OpenWindows implements both the 2-D and 3-D OPEN LOOK GUI standard.

OpenWindows is a component of Solaris 2.5. The OPEN LOOK components indicated below implement the OPEN LOOK GUI.

- OPEN LOOK Intrinsics Toolkit (OLIT): The OPEN LOOK Intrinsics Toolkit was created by building a set of widgets for the X Toolkit Intrinsics (Xt) that conform to the OPEN LOOK GUI specification. The OPEN LOOK Intrinsics Toolkit API matches the X11 Release 4 Xt Intrinsics programming interface. For Solaris 2.5, OLIT is Level 2 compliant with certain exceptions.
- XView Toolkit (X11-based Visual/Integrated Environment for Workstations): XView is an X11 toolkit for building applications. The XView API is based upon Xlib, the lowest level of programming available to the X window system programmer. XView implements the OPEN LOOK GUI. For Solaris 2.5, XView is Level 2 compliant with certain exceptions.

- OpenWindows DeskSet: OpenWindows includes a set of OPEN LOOK applications that are collectively known as the DeskSet environment. All of the DeskSet applications support the OPEN LOOK model of dragging and dropping objects. The File Manager DeskSet tool is required by the OPEN LOOK standard for Level 2 compliance; it represents files (including directories and applications) with glyphs. The OpenWindows DeskSet File Manager program fulfills the Level 2 compliance requirement.

OPEN LOOK Specification and Related References

The specifications listed below address OPEN LOOK. They are published by Addison-Wesley.

- *OPEN LOOK Graphical User Interface Functional Specification* (ISBN 0-201-52365-5)
- *OPEN LOOK Graphical User Interface Application Style Guidelines* (ISBN 0-201-52364-7)

Licensing the OPEN LOOK Trademark

OPEN LOOK is a trademark of UNIX System Laboratories (USL), a wholly owned subsidiary of Novell, Inc. The Trademark License Agreement is the contract entered into by OPEN LOOK trademark applicants and USL. As a developer of OPEN LOOK applications, you may wish to license the OPEN LOOK trademark. USL has developed a trademark agreement as a formality to protect the trademark. To obtain the use of the OPEN LOOK trademark, follow the recommendations described below; no payment is required.

- For information on how to develop an OPEN LOOK application, refer to the *OPEN LOOK Graphical User Interface Functional Specification* that is delivered with OpenWindows.
- To receive your “OPEN LOOK Graphical User Interface Trademark License Agreement” forms, call 1 (800) 828-UNIX. If you are a UNIX system licensee, ask for your account representative. Otherwise, a sales associate will handle your request. After an authorized representative of your company signs the agreement, send it to USL at the following address:

UNIX System Laboratories
Attention: Sales Associate (or the name of your account representative)
PO Box 25000
Greensboro, NC 27420-5000.

Within 30 days, USL will send you an executed agreement authorizing you to use the OPEN LOOK trademark on your software.

X Window System, Version 11 (X11)

The X Window System, Version 11 (X11), developed by the Massachusetts Institute of Technology (MIT) X Consortium includes the following specifications: the Xlib C Language Interface (Xlib), the X Toolkit Intrinsics C Language Interface (Xt), and the Bitmap Distribution Format 2.1 (BDF).

X11 is a network-based protocol. A client application can run on the same or different system from the server that controls the display. In this server-client model, the application (which may run on one machine) is referred to as the window client. The system on which the user-interface is displayed may be a different machine and is referred to as the display host or window server.

Window systems are usually based on a pixel imaging model or a stencil / paint imaging model. The imaging model layer of the windows architecture controls how the window system accesses the display. X11 uses a pixel-based (raster) model in which images are viewed as rectangular areas of device-dependent pixels.

The Xlib library routines communicate with the X11 server via the X protocol. Xlib is the lowest-level C language application programming interface (API) to the X protocol.

The main task of Xlib is to translate C data structures and procedures into X protocol events; it sends them off and receives protocol packets in return that are unpacked into C data structures. Xlib provides full access to the capabilities of the X protocol but does little to make programming easier. It handles the interface between an application and the network and includes some optimizations that encourage efficient network usage.

Because application development at the Xlib level can be tedious, MIT developed the X toolkit, Xt. The designers of Xt were aware that the toolkit would need to support a variety of graphical user interface standards. For this reason, Xt was divided into two portions. The first portion is a prebuilt set of user interface components known as widgets. The second portion is the programmer interface for manipulating widgets, known as intrinsics.

Although it is device-independent, X11 allows an application to tailor itself to the hardware on which it is run.

PostScript Language

The PostScript™ language, from Adobe Systems Inc., is the modern standard for electronic printing. The first edition of the *PostScript Language Reference Manual*, published in 1985 by Addison and Wesley, established PostScript Level One. Today, PostScript is supported as a standard by all major computer, printer, and imagesetter vendors.

Numerous extensions were requested by the industry, so, in 1990, the second edition of the reference manual was published. It describes three major extensions to PostScript Level One: (1) An extension to deal with color output, (2) A composite font model, mainly used for very large fonts (for example, Asian languages) and (3) A set of extensions for screen output, called the Display PostScript™ system, or DPS. DPS displays graphical information on the computer screen with the same imaging model and PostScript language that are the standards for printers and typesetters.

These major extensions and a large number of minor ones comprise PostScript Level Two, often referred to as PS2 or PSL2.

Solaris Compliance With X11

OpenWindows consists of the OpenWindows server, the Display PostScript™ (DPS) extension support, the OpenFonts™ Technology, the OPEN LOOK window manager (olwm), the XView Toolkit, the OPEN LOOK Intrinsics Toolkit (OLIT), the DeskSet™ tools, and demonstration applications.

The OpenWindows server and the associated X libraries, which include Xlib and Xt, are compliant with X11, Release 5.

The OPEN LOOK Intrinsics Toolkit API is an implementation of MIT's Xt toolkit with an OPEN LOOK widget set. AT&T created the OPEN LOOK Intrinsics Toolkit by building a set of widgets for Xt that conform to the OPEN LOOK GUI specification.

The XView toolkit (X Window System-based Visual/Integrated Environment for Workstations) is a C-language toolkit providing a rich set of components for building applications. Like the Intrinsics, XView is built on Xlib. SunSoft has made the source code to the XView Toolkit freely available. It is shipped as part of the standard MIT X distribution and with UNIX System V Release 4.

OpenWindows, through the OPEN LOOK window manager, fully supports the X11 Inter-Client Communications Conventions (ICCC) as defined in X11 Release 5. The ICCC manual provides basic policy intentionally omitted from X itself, such as rules for transferring data between applications, transfer of keyboard focus, layout schemes, colormap installation and other features.

Solaris Compliance with PostScript

The OpenWindows server is a complete implementation of PostScript Level Two. The Display PostScript system is implemented as an extension to the X Window System and includes the following enhancements:

- Support for F3 Latin and Asian fonts
- Support for obtaining prescaled bitmap font formats from X11 font code

X11 Specification and Related Publications

The first publication listed below defines the X11 protocol specification; it is also defined in subsequent supplements supplied with X11 Release 5.

- *X Window System Third Edition*, Schiefler & Gettys, Digital Press, 1992
- *XView Programming Manual*, O'Reilly & Associates, Inc., 1989
- *XView Reference Manual*, O'Reilly & Associates, Inc.
- *PostScript Language Reference Manual, Second Edition*, Addison-Wesley
- *XView Developer's Notes*, Sun Microsystems, Inc.
- *OpenWindows Desktop Reference Manual*, Sun Microsystems, Inc.
- *Desktop Integration Guide*, Sun Microsystems, Inc.

Common Desktop Environment 4

This chapter introduces the Common Desktop Environment (CDE), a graphical user interface and development environment that debuts with Solaris 2.5. This chapter provides an overview of CDE's development history, discusses the compliance of CDE to industry standards and specifications and outlines current efforts at standardizing the CDE components.

Common Desktop Environment

In March of 1993, Sun, Hewlett–Packard, IBM and Novell announced an agreement to develop a graphical user interface that would bring a consistent look and feel to major UNIX system-based workstations and desktop computers. From the start, the CDE development effort was guided by one goal: to make UNIX easier-to-use for end users and application developers.

The result of this joint development effort is the Common Desktop Environment or "CDE." CDE debuts with Solaris 2.5, and along with OpenWindows, is one of two desktops packaged with this release. Over time, CDE will emerge as the standard desktop for Sun, Hewlett–Packard, IBM, Novell and many others in the UNIX workstation market.

CDE includes a desktop server, a Session Manager, a Window Manager (based on HP's Visual User Environment), and numerous desktop utilities, (based on SunSoft's OPEN LOOK and DeskSet tools).

Developers, End Users, and CDE

Because CDE provides a consistent computing environment across major UNIX platforms, end users will have less trouble moving between different machines. CDE also aids application development by supplying a single, standard set of programming interfaces for any conforming Sun, HP, IBM, and Novell platform. A single API allows developers to create applications that will be consistent in appearance and behavior across CDE-compliant systems.

The CDE development environment is based on the X11R5 server and produces applications with a look and feel based on the Open Software Foundation's Motif 1.2 specification.

Standardizing the Common Desktop Environment

Recognizing the development and procurement needs of their customers, the CDE development partners have pledged that the desktop will comply with industry standards wherever possible. Where a standard does not yet exist, CDE specifications are being provided to recognized standards bodies for consideration as new standards.

The remaining sections of this chapter detail the compliance of various CDE components to major standards and specifications; additionally, it provides an overview of the cooperative efforts of the CDE partners and major standards bodies in the development of new CDE-related standards.

Open Software Foundation/Motif

The Motif graphical user interface was developed by the Open Software Foundation (OSF) and based on work by Hewlett–Packard and Digital Equipment Corp. Motif uses windows and menus with common graphic symbols to provide an intuitive, consistent environment across a variety of platforms that support the X Window System, X11R5.

Motif 1.2 consists of a user interface toolkit, a user interface language, a window manager and a single application programmers interface. Motif 1.2 serves as the base graphical user interface specification for the Common Desktop Environment.

OSF/Motif Application Certification

OSF/Motif's Certification and Trademark Program offers application developers a method to communicate compliance with OSF/Motif. A Motif–compliant application must be developed with the Motif toolkit and run in a Motif–compliant environment.

The *OSF/Motif Style Guide* describes the way applications should "behave" or interact with users and offers the basis for application certification. Developers that follow the guidelines presented in the *Style Guide* are assured that their applications will emulate the Motif look. Applications that pass the certification process are granted use of a statement signifying compliance to the *OSF/Motif Style Guide*.

Note – OSF has announced submission of the Motif API specification to X/Open for licensing. Once branding is awarded, (expected in late 1995), the OSF/Motif certification program will be overseen by X/Open. Until that time, OSF will continue to certify implementations.

CDE and Motif

CDE's development toolkit is an enhanced version of Motif 1.2 that has been dubbed "CDE Motif." CDE Motif is based on Motif 1.2.3, but extends the Motif widget library and contains some new features. CDE Motif shares source and binary compatibility with Motif, meaning that existing Motif 1.2 applications will compile and run on CDE platforms without modification.

The CDE Motif API is based on the Motif 1.2 toolkit API and is compliant with the IEEE Std. 1295–1993 API specification.

Porting Motif Applications to CDE/Developing New CDE Applications

Developers wishing to port an existing Motif 1.2 application to CDE or develop a new CDE–compliant application should follow the guidelines outlined in *The Common Desktop Environment: Style Guide and Certification Checklist*. It defines the guidelines that allow an existing Motif application to integrate well with CDE.

Note – Because Motif 1.2 defines only the basic behavior for applications and widgets and not for a desktop, *The Common Desktop Environment: Style Guide and Certification Checklist* supplements and extends the *OSF/Motif Style Guide*. For this reason, developers wishing to write a CDE conforming application should follow the guidelines presented in both guides.

Developers porting application to the desktop or developing new CDE applications can choose from three levels of application integration:

- Level 1– Minimal: This level requires that the application appear under the desktop's Application Manager folder and be launchable from the application's data file icon.
- Level 2– Recommended: This level requires that the application interact with the desktop's components.
- Level 3– Optional: This level provides an integration checklist of desktop components with which applications may optionally interact.

Certifying CDE Applications

Compliance with CDE interface guidelines is voluntary and self-regulated; there is no formal certification process. Applications that meet all the required guidelines of the *CDE Style Guide and Certification Checklist* and the *OSF/Motif Style Guide* can be considered CDE-compliant.

For a discussion of issues that can affect application portability between different platforms that support CDE, see the *Common Desktop Environment: Programmer's Overview.*

Solaris Compliance With OSF/Motif

At the time of this book's publication, the OSF/Motif certification process was not yet complete. However, SunSoft fully expects Solaris to be certified as a conforming implementation of Motif 1.2 for its 2.5 release.

OSF/Motif Specifications and Related References

For more information on the Motif 1.2 toolkit and for guidelines on developing Motif 1.2 applications, see the following OSF publications:

OSF/Motif Style Guide, Release 1.2, PTR Prentice Hall

OSF/Motif Programmer's Reference, Release 1.2, PTR Prentice Hall

OSF/Motif Programmer's Guide, Release 1.2, PTR Prentice Hall

CDE/Motif Specifications and Related References

For more information on the CDE Motif toolkit and for guidelines for developing CDE Motif applications, see the following publications:

Common Desktop Environment: Programmer's Overview

Common Desktop Environment: Programmer's Guide

Common Desktop Environment Style Guide and Certification Checklist

The X Window System

The X Window System, Version 11 (X11), includes the following specifications: the Xlib C Language Interface (Xlib), the X Toolkit Intrinsics C Language Interface (Xt), and the Bitmap Distribution Format 2.1 (BDF).

X11 is a network-based protocol. A client application can run on the same or different system from the server that controls the display. In this server-client model, the application (which may run on one machine) is referred to as the window client. The system on which the user-interface is displayed may be a different machine and is referred to as the display host or window server.

The X library routines or “xlib” provide a standard programmer's interface to the X Window System. Xlib is a standard set of C language routines that developers can use to program basic graphics functions, and that automatically produce the corresponding X protocol. Xlib communicates with the X11 server via the X protocol.

The main task of Xlib is to translate C data structures and procedures into X protocol events; it sends them off and receives protocol packets in return that are unpacked into C data structures. Xlib provides full access to the capabilities of the X protocol but does little to make programming easier. It handles the interface between an application and the network and includes some optimizations that encourage efficient network usage.

Because application development at the Xlib level can be tedious, MIT developed the X toolkit, Xt. The designers of Xt were aware that the toolkit would need to support a variety of graphical user interface standards. For this reason, Xt was divided into two portions. The first portion is a prebuilt set of user interface components known as widgets. The second portion is the programmer interface for manipulating widgets, known as intrinsics.

CDE Compliance with X11

The Common Desktop Environment consists of the desktop server, the Session Manager, the Window Manager, the CDE/Motif toolkit, the desktop utilities and the Display PostScript™ (DPS) extension support.

The desktop server and the associated X libraries, which include Xlib and Xt, are fully compliant with X11, Release 5.

The Session Manager fully supports the X11 Inter-Client Communications Conventions (ICCCM) 1.1 Session Management protocol. The Session Manager preserves the state of applications at logout.

The Window Manager, which is based on the Motif 1.2.3 window manager with workspace extensions, is ICCCM compliant.

The CDE/Motif toolkit is an implementation of OSF/Motif's 1.2.3 toolkit with extended functionality. The CDE/Motif toolkit is based on X11 Intrinsics, which is specified as a U.S. federal procurement standard.

Solaris Compliance with PostScript

The Common Desktop Environment's server is a complete implementation of PostScript Level Two. The Display PostScript system is implemented as an extension to the X Window System and includes the following enhancements:

- Support for F3 Latin and Asian fonts
- Support for obtaining prescaled bitmap font formats from X11 font code

X11 Specification and Related Publications

The first publication listed below defines the X11 protocol specification; it is also defined in subsequent supplements supplied with X11 Release 5.

- *X Window System Third Edition*, Scheifler & Gettys, Digital Press, 1992
- *PostScript Language Reference Manual, Second Edition*, Addison-Wesley

Future Standardization

A number of CDE services and applications are currently targeted for standardization. At the time of this book's publication, the CDE standardization efforts were ongoing and not yet complete.

This section discusses the CDE standardization efforts, identifies the standards bodies working with the CDE development partners and identifies the CDE components targeted for standardization.

X/Open CDE Specification

Specifications for a broad range of CDE components have been submitted to X/Open for consideration as standards. Representatives from Sun, Hewlett–Packard, IBM, and Novell are working closely with X/Open in guiding the emergence of the new X/Open CDE standards.

CDE components that are branded by X/Open will become part of X/Open's Common Applications Environment (CAE), which covers the standards that are required to support open systems.

Among the CDE components being submitted for incorporation into the Portability Guide are the CDE Motif toolkit API, the Session Manager, the data interchange "drag and drop" protocols, and the Application Help Developer's Kit.

At present, X/Open has completed a "fast track" review of the submitted CDE specifications. If approved, publication of the final standard is expected by Spring 1995. A formal branding and certification program is expected to be in place by late 1995.

X/Open Calendaring and Scheduling API

The X/Open Calendaring and Scheduling Application Program Interface (CSA API) is a high-level interface that facilitates the development of calendar-enabled applications.

The X/Open CSA API is based upon work done by the X.400 Application Programming Interface Association (XAPIA). It defines a set of high-level functions that are available to calendar enabled applications, including adding, deleting, modifying, and reading a calendar and the calendar entries.

The interface supports searches for free time intervals within a calendar through a generic definition of capabilities for calendaring and scheduling. The X/Open CSA API features a single function set that helps minimize the number of function calls needed to manage multiple types of calendar entries.

In November 1994, a preliminary specification for the Calendaring and Scheduling API was published by X/Open. The final specification is expected by Spring 1995. The X/Open branding and certification program is expected to be in place by late 1995. CDE's calendaring and scheduling service is expected to fully comply with the final X/Open specification.

X/Open and XPG3 5

The X/Open consortium was established to make multivendor open systems a practical reality. X/Open takes existing standard interfaces and adapts them to the specifics of open systems. These interfaces comprise what is known as the Common Applications Environment (CAE) and are documented in the X/Open Portability Guide.

This chapter discusses the compliance of Solaris 2.5 to the programming interface specifications detailed in the X/Open Portability Guide, Issue 3 (XPG3).

The X/Open Portability Guide, Issue 3

In 1988, X/Open published the X/Open Portability Guide Issue 3, commonly referred to as "XPG3." It is a collection of seven volumes that includes the interfaces specified in the IEEE 1003.1-1988 POSIX standard.

Adherence to the programming interface specifications contained in XPG3 ensures application portability at the source code level. Compliance with these interfaces is determined through an extensive set of conformance tests and is assured through the X/Open branding process which entitles a product to bear the X/Open trademark.

Note – In 1992, X/Open published Issue 4 of the Portability Guide, (XPG4). It retains compliance to the IEEE 1003.1-1988 standard but is extended to the ISO/IEC updated POSIX.1 standard and the ISO/IEC C language standard. While XPG3 is still available and systems and components can still be branded to XPG3, XPG4 offers significant additional capability. For more information on XPG4, see Chapter 6, "X/Open and XPG4."

This chapter identifies Solaris 2.5 as a conforming implementation of XPG3, and displays the XPG3 Base brand trademark. It also presents the X/Open Conformance Statement, which documents Solaris' compliance to the programming interface specifications of the X/Open Portability Guide, Issue 3.

The X/Open Brand Trademark

X/Open provides a verification and branding program that developers can use to show that their products are X/Open compliant. Sun Microsystems has been a strong supporter of the X/Open branding process since its inception.

Components of the Common Applications Environment are categorized into three levels: BASE, PLUS, and OPTIONS. A system that provides all of the BASE components is awarded an XPG3 BASE profile trademark. A system that implements all of the BASE and PLUS components may bear the XPG3 PLUS profile trademark.

Figure 5-1 The XPG3 Base Brand Logo

X/Open
BASE XPG3

Solaris has earned the XPG3 BASE brand. Solaris products and software products from independent software vendors that have received XPG3 branding are described below:

- **Window Management (OpenWindows)**—The Solaris window system, which supports the OPEN LOOK Graphical User Interface, has earned the XPG3 brand by implementing the programmer's interface to the X Window System. OpenWindows supports the Window Management component of the X/Open PLUS level.

- **Commands and Utilities**—X/Open's specification of standard interfaces for utilities allows for portable shell scripts. Solaris meets the Commands and Utilities component requirements of the BASE system.
- **ProCompiler™ C 2.0.1**— The ProCompiler for Solaris for x86 is fully conformant with the ANSI/ISO Standard for C. It has passed X/Open verification test suite VSX3 and meets the C Language component requirements of the BASE level.
- **SPARCompiler C 2.0.1**—The SPARCompiler for the C programming language based on Common Usage C has passed X/Open verification test suite VSX4.2.4 and meets the C Language component requirements of the BASE level.
- **Sun FORTRAN 3.0**—Sun's compiler for the FORTRAN programming language is fully compliant with the definition in the American National Standards Institute (ANSI) document and carries the XPG3 brand when run on Solaris.
- **Sun Pascal 3.0.1**—Sun's compiler for the Pascal programming language is fully compliant with the ISO standard and carries the XPG3 brand when run on Solaris.
- **Magnetic Media (Source Code Transfer)**—Sun conforms to X/Open's specifications for transferring source code between machines with compatible media and facilitating the transfer of source code in machine-readable form. Solaris supports the Source Code Transfer component of the X/Open OPTIONS level.
- **Inter-Process Communication**—Sun supports X/Open's specifications for interfaces providing message queue, semaphore, and shared memory facilities for communication and synchronization between processes. The SunOS operating system fulfills the requirements of the Inter-Process Communications component of the X/Open OPTIONS level.
- **Terminal Interfaces (XSI Curses Interface)**—The XSI Curses Interface meets X/Open's specifications for providing a generic terminal interface that is independent of terminal hardware or connection methods for updating screens on character-oriented and block-oriented terminals. Solaris systems fulfill the requirements of the Terminal Interfaces component of the X/Open OPTIONS level.

5

X/Open Conformance Statement for Solaris

The remaining pages of this chapter feature the X/Open Conformance Statement for Solaris.

X/Open Conformance Statement

X/OPEN Conformance Statement Questionnaire

Chapter 2: Internationalized System Calls and Libraries

Product Identification

Product Identification	Solaris
Version/Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

VSX Test Suite Release	VSX 4.3.2
Testing Agency Name	Mindcraft, Inc.
Address	410 Cambridge Avenue Palo Alto, California 94306

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC

SPARC running Solaris 2.5. Installation procedures are provided in SPARC: Installing Solaris Software

To reproduce the test environment, follow these steps:

1. Edit the `/etc/saf/zsmon/_pmtab` file to turn off the ttysoftcarrier detect:

   Change the `ttya` and `ttyb` fields from `:y:` to `:n:`. (The colons (:) act as field separators).

2. Verify that the ttymodes settings in the `/kernel/drv/options.conf` file are set to:

```
2502:1805:bd:8a3b:3:1c:7f:15:4:0:0:0:11:13:la:19:12:f:17:16
```

3. Disable `ypbind` to allow rebooting of the system:

   a. cd `/usr/lib/netsvc/yp`

   b. `mv ypbind ypbind-`

4. Set the `eeprom` variables that affect the `tty`:

   a. On the keyboard, hit `STOP-A` to display the `prom` prompt.

   b. At the prompt, execute the following steps:

```
setenv ttya-ignore-cd false
setenv ttyb-ignore-cd false
setenv ttya-rts-dtr-off false
setenv ttyb-rts-dtr-off false
```

5. Reboot the system

Note – When installing Solaris, set the time zone by selecting a time zone format that conforms to the POSIX.1 format for TZ defined on page 152 and page 153 of the IEEE Std. 1003.1–1990.

x86

x86 running Solaris 2.5. Installation procedures are provided in x86: Installing Solaris Software.

To reproduce the test environment, follow these steps:

1. Become `root`.

2. Ensure the correct serial port links:

- `/dev/ttya` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/term/a` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/tty00` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/ttyb` should be a link to `/devices/isa/asy@2f8,0:a`
- `/dev/term/b` should be a link to `/devices/isa/asy@2f8,0:a`
- `/dev/tty01` should be a link to `/devices/isa/asy@2f8,0:b`

   a. If the `/dev/tty01` link is missing, perform the following:

- Edit `/kernel/drv/asy.conf` and uncomment the COM2 entry
- `# touch /reconfigure`

3. Set the correct serial port permissions:

- `# chmod 666 /devices/eisa/asy*`

4. Turn off the `ttysoftcarrier` detect:
   Using an editor such as `vi`, in the `/etc/saf/zsmon/_pmtab` file, change the next to last field for both the `ttya` entry and the `ttyb` entry from `y` to `n` (the colon (`:`) acts as the field separator):

- `# vi /etc/saf/zsmon/_pmtab`

5. Reboot the system.

Note – When installing Solaris, set the time zone by selecting a time zone format that conforms to the POSIX.1 format for TZ defined on page 152 and page 153 of the IEEE Std. 1003.1–1990.

Temporary Waivers

List below references to any temporary waivers granted by X/Open in respect to minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

Section 2.1 General Attributes

2.1.1 POSIX.1 Supported Features

Question 1: *Which of the following options, specified in the* **<unistd.h>** *header fileSCC are available on the system?*

Answer:

Macro Name	Meaning	Provided
_POSIX_CHOWN_RESTRICTED	The use of **chown()** is restricted	Variable
_POSIX_JOB_CONTROL	Job Control option	Yes
_POSIX_NO_TRUNC	Long pathname components generate an error	Variable
_POSIX_SAVED_IDS	Effective user and group IDs are saved	Yes
_POSIX_VDISABLE	Terminal special characters can be disabled	Variable

When native SunOS file systems and terminal drivers are used, _POSIX_CHOWN_RESTRICTED is supported, _POSIX_NO_TRUNC is supported, and _POSIX_VDISABLE has the value ‘0’, or ‘\0’ in C language source. When other file system types are used, such as through NFS, or terminal drivers from third party vendors, the results may vary and can be queried using `pathconf()` and `fpathconf()`.

Rationale

For an X/Open conforming implementation, the `_POSIX_SAVED_IDS` option must be provided. The other options may or may not be provided. The provision of the file system related options can vary within a system. For example, a system which has traditionally supported both System V and BSD type file systems may provide a mechanism whereby the option is enforced for certain files or processes but not for others. This technique can be used to achieve a degree of backwards compatibility that would not otherwise be possible.

Reference

XPG3 Volume 2, page 579-**<unistd.h>**

2.1.2 C Standard

Question 2: *Does the implementation only support* **Common Usage C** *or also support* **ANSI C Standard** *interface definitions?*

Answer:

Both Common Usage C and ANSI C are provided.

Rationale

The POSIX.1 standard allows for a conforming system to support either Common Usage C or ANSI C Standard interface definitions. The XPG is based on a Common Usage C definition but does not prohibit an ANSI C implementation. A Common Usage C definition must provide function declarations for the C language functions in the XPG as well as providing function semantics that conform to the XPG. An ANSI C Standard interface must provide function prototypes and ANSI C semantics as well as providing XPG semantics. There are no known areas of contradiction between the ANSI C and the XPG semantics.

Reference

XPG3 Volume 2, page 12 - *The Compilation Environment*

2.1.3 Limit Values

Question 3: *What are the values associated with the following limits specified in the* <**limits.h**> *header file?*

Answer:

Macro Name	Meaning	Minimum	Maximum
ARG_MAX	Max length of argument list and environment data	4096	1048320
CHILD_MAX	Max number of processes per user ID	6	See note
LINK_MAX	Max number of links to a single file	8	32767
MAX_CANON	Max bytes in a terminal canonical input line	255	256

MAX_INPUT	Max bytes in a terminal input queue	255	512
NAME_MAX	Max characters in a filename	14	See note
OPEN_MAX	Max number of files open in a process	16	See note
PASS_MAX	Max significant characters in a password	8	8
PATH_MAX	Max characters in a pathname	255	See note
PIPE_BUF	Max bytes in an atomic write to a pipe	512	5120
NGROUPS_MAX	Max number of supplementary group IDs	0	16
TMP_MAX	Max number of unique temporary file names	17576	17576

Notes:
CHILD_MAX depends on how the system kernel is configured.

The maximum values for NAME_MAX and PATH_MAX vary depending on the file system type, but always provide at least the minimum requirement. The most common values are 255 for NAME_MAX and 1024 for PATH_MAX. Values for a specific path are available using `pathconf()`.

OPEN_MAX defaults to 64, but users can increase or decrease this value using routines not specified by POSIX.1 or XPG3.

Rationale

Each of these limits can vary within bounds set by the X/Open Portability Guide. The minimum value that a limit can take on any X/Open conforming system is given in the corresponding **_POSIX_** value. A specific conforming implementation may provide a higher minimum value than this and the maximum value that it provides can differ from the minimum. Some conforming implementations may provide a potentially infinite value as the maximum, in which case the value is considered to be indeterminate. The minimum value must always be definitive since the **_POSIX_** value provides a known lower bound for the range of possible values.

Reference

XPG3 Volume 2, page 538 - <**limits.h**>

Question 4: *What are the values associated with the following constants specified in the <***limits.h***> header file?*

Answer:

Macro Name	Meaning	Value
CHAR_BIT	Number of bits in a char	8
LONG_BIT	Number of bits in a long	32
WORD_BIT	Number of bits in a word	32
DBL_DIG	Digits of precision of a double	15
DBL_MAX	Maximum decimal value of a double	1.7976931348623157E+308
FLT_DIG	Digits of precision of a float	6
FLT_MAX	Maximum decimal value of a float	3.40282347E+38

Rationale

This set of constants provides useful information regarding the underlying architecture of the implementation.

Reference

XPG3 Volume 2, page 537 - <**limits.h**>

2.1.4 Error Conditions

Question 5: *Which of the following optional errors listed in the XPG are detected in the circumstances specified?*

Answer:

Function	Error	Detected
access()	EINVAL†	Yes
	ETXTBSY	No
atof()	ERANGE	Yes
atoi()	ERANGE	Yes

(continued)

Function	Error	Detected
atol()	ERANGE	Yes
cfsetispeed()	EINVAL	No
cfsetospeed()	EINVAL	No
chmod()	EINVAL	No
chown()	EINVAL†	Yes
closedir()	EBADF†	Yes
exec	ENOMEM†	Yes
	ETXTBSY	No
fcntl()	EDEADLK†	Yes
fdopen()	EBADF	No
	EINVAL	No
feof()	EBADF	No
ferror()	EBADF	No
fileno()	EBADF	No
fopen()	EINVAL	No
	ETXTBSY	No
freopen()	EINVAL	No
	ETXTBSY	No
fork()	ENOMEM	Yes
fseek()	EINVAL	Yes
ftw()	EINVAL	No
getcwd()	EACCES†	Yes
isatty()	EBADF	No
	ENOTTY	No

(continued)

Function	Error	Detected
open()	EINVAL	No
	ETXTBSY	No
opendir()	EMFILE†	Yes
	ENFILE†	Yes
pathconf()	EACCES†	Yes
	EINVAL†	Yes
	ENAMETOOLONG†	Yes
	ENOENT†	Yes
	ENOTDIR†	Yes
fpathconf()	EBADF†	Yes
	EINVAL†	Yes
printf()	EINVAL	No
readdir()	EBADF†	Yes
rename()	ETXTBSY	No
scanf()	EINVAL	No
setvbuf()	EBADF	No
sigaddset()	EINVAL†	Yes
sigdelset()	EINVAL†	Yes
sigismember()	EINVAL†	Yes
strcoll()	EINVAL	No
strerror()	EINVAL	No
strtol()	EINVAL	Yes
	ERANGE	Yes
strxfrm()	EINVAL	No
unlink()	ETXTBSY	No

Rationale

Each of the above error conditions is marked as optional in the XPG and an implementation may return this error in the circumstances specified or may not provide the error indication. Those items marked with a † are also considered to be optional error conditions in POSIX.1. The EINVAL error condition for the three functions **sigaddset()**, **sigdelset()**, and **sigismember()** are mandated in the XPG but are considered optional in POSIX.1. An X/Open conforming implementation will always produce these errors, but a POSIX.1 conforming implementation may not.

Reference

XPG3 Volume 2, page 32– Error Numbers.

2.1.5 Mathematical Interfaces

Question 6: *What format of floating point numbers is supported by this implementation?*

Answer:

IEEE floating point format is supported.

Rationale

Most implementations support IEEE floating point format either in hardware or software. Some implementations support other formats with different exponent and mantissa accuracy. These differences need to be defined.

Question 7: *Is* **long double** *form supported and what precision is associated with this form?*

Answer:

Long double uses 16 bytes. The low order 112 bits are used to hold the mantissa, the next 15 bits hold the exponent, and the high order bit is used as the sign bit.

Rationale

The long double format can vary both in length and precision. If it is supported, other than as a synonym for double, the format needs to be described.

Reference

XPG3 Volume 2, page 328 - printf()
XPG3 Volume 2, page 362 - scanf()

2.1.6 Data Encryption

Question 8: *Are the optional data encryption interfaces provided?*

Answer:

crypt()	Yes
encrypt()	Yes (Decryption capabilities not provided to areas restricted by U.S Export Law.)
setkey()	Yes

Rationale

Normally an implementation will either provide all three of these routines or will provide none of them at all. If the routines are not provided, then the implementation must provide a dummy interface which always raises an ENOSYS error condition.

It is also possible that the implementation of the **encrypt()** function may be affected by export restrictions, in which case, the restrictions should be documented here.

For example, historical implementations have supplied all three of the routines outside the USA, but due to export restrictions on the decoding algorithm, a dummy version of **encrypt( )** is provided that does encoding, but not decoding.

Reference

XPG3 Volume 2, page 3 - *Status of Interfaces*

Section 2.2 Process Handling

2.2.1 Process Generation

Question 9: *Which file types (regular, directory, FIFO, special etc.) are considered to be executable?*

Answer:

Only regular files may be executed.

Rationale

The EACCES error associated with **exec** functions occurs in circumstances when the implementation does not support execution of files of the type specified. A list of these file types needs to be provided.

Reference

XPG3 Volume 2, page 129 -exec

2.2.2 Process Termination

Question 10: *Is the* SIGCHLD *signal sent to the parent process when a child exits?*

Answer:

Yes

Rationale

Some systems support the sending of SIGCHLD in these circumstances. This is mandatory if job control is supported.

Reference

XPG3 Volume 2, page 132 -exit()

2.2.3 Process Environment

Question 11: *Is the* **setpgid()** *interface provided?*

Answer:

Yes

Rationale

This interface is mandatory on systems which support job control and may be provided on other systems.

Reference

XPG3 Volume 2, page 3 - *Status of Interfaces*

Section 2.3 File Handling

2.3.1 Access Control

Question 12: *What file access control mechanisms does the implementation provide?*

Answer:

There is no additional access or optional file control mechanism.

Rationale

The XPG (and POSIX) allow an implementation to provide either additional or alternate file access control mechanisms other than the standard access control mechanism. The document should either describe or provide a reference to the details of alternate or additional access mechanisms. In particular, the method by which an application can execute (using standard file access control) should be explained, and details of the changes required to utilize alternate or additional access mechanisms should be given.

Reference
XPG3 Volume 2, page 16 - *File Access Permissions*

2.3.2 Files and Directories

Question 13: *Are any extended security controls implemented that could cause* fstat() *or* stat() *to fail?*

Answer:
No

Rationale
The XPG notes that there could be an interaction between extended security controls and the success of **fstat()** and **stat()**. This would suggest that an implementation can allow access to a file but not allow the process to gain information about the status of the file.

Reference
XPG3 Volume 2, page 478 -tempnam()

2.3.3 Formatting Interfaces

Question 14: *Is the L modifier to* **printf()** *and* **scanf()** *supported in this implementation?*

Answer:
Yes

Rationale
The XPG notes that the L modifier, which is exactly equivalent to the l modifier when the implementation does not differentiate between double and long double, is not supported on all systems and is only included for compatibility with ANSI C.

Reference
XPG3 Volume 2, page 328 - printf()
XPG3 Volume 2, page 362 - scanf()

Question 15: *Does the* **printf()** *function produce character string representations for Infinity and NaN to represent the respective special double precision values?*

Answer:
Yes

Rationale
This behavior is often provided on systems with mathematical functions that produce these results.

Reference
XPG3 Volume 2, page 331 - printf()

Section 2.4 General Terminal Interface

2.4.1 Interfaces Supported

Question 16: *Are the following terminal control interfaces provided?*

tcgetpgrp()	tcsetpgrp()

Answer:
Yes

Rationale
These interfaces are mandatory for implementations that support **job control**. Implementations that do not support **job control** may either always return the error indication [ENOSYS] or may provide the interface with the behavior specified for an implementation that supports **job control**. The latter case is useful for implementations that support only part of the **job control** specifications.

Reference
XPG3 Volume 2, page 471 - tcgetpgrp
XPG3 Volume 2, page 475 - tcsetpgrp

Section 2.5 Internationalized System Interfaces

2.5.1 Codesets

Question 17: *Does the implementation support the* **ISO 8859-1:1987** *codeset for data transmission?*

Answer:
Yes

Rationale

The XPG defines the ISO 8859-1:1987 as the major Western European transmission codeset and also recommends its use as the corresponding internal codeset.

Reference

XPG3 Volume 3, page 19 - *Character Codesets and Text Transfer*

Question 18: *Does the implementation use the* **ISO 8859-1:1987** *as its internal codeset?*

Answer:

Yes

Rationale

The XPG defines the ISO 8859-1:1987 as the major Western European transmission codeset and also recommends its use as the corresponding internal codeset.

Reference

XPG3 Volume 3, page 19 - *Character Codesets and Text Transfer*

2.5.2 Regular Expression Interfaces

Question 19: *What form of regular expression syntax is supported by the* **regexp()** *interface?*

Answer:

Simple regular expression

Rationale

The **regexp()** interface may support either the simple regular expression or the simple internationalized regular expression syntax as defined in the XPG3 Volume 3 - *Supplementary Definitions.*

Reference

XPG3 Volume 3, pages 49-51 - *Regular Expressions*

Chapter 3: Commands and Utilities

Product Identification

Product Identification	Solaris
Version / Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

None

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC and x86, running Solaris. Installation procedures are provided in *SPARC: Installing Solaris Software or x86: Installing Solaris Software*.

Conformance Expectation

Volume 1 of XPG3 recognizes that convergence of implementations towards a common specification for commands and utilities is not yet complete and therefore does not require a vendor to supply all of the commands and utilities (and individual options) specified in XPG3.

This chapter explicitly identifies those commands and utilities not supplied by the vendor and any supplied that do not conform to the published specification. (Reference: XPG3 Volume 1, page 1.)

Section 3.1 Basic Utilities

3.1.1 Supported Commands

Question 1: *Which of the basic utilities (non-development utilities) defined in the XPG are not provided with the implementation?*

Answer:
All the defined utilities are provided.

Rationale
The XPG Volume 1 states that “this volume in its current form is useful only as a guide to portability, but it is not possible to precisely define or test conformance to it.” This question determines whether or not the implementation provides a command of the name specified in the XPG; it does not attempt to determine whether it supports the semantics of that command. The (optional) development utilities are excluded from this question and are dealt with in the next section of the questionnaire.

Reference
XPG3 Volume 1, page 1 - *Introduction*

3.1.2 Command Behavior

Question 2: *In what ways do the commands provided by the implementation behave differently from the specifications contained in the XPG?*

Answer:
The -n option to ps is not supported.

Rationale
This question provides a greater degree of granularity than the previous question, requiring the semantic differences associated with the commands to be specified. Again, the question relates to the basic utilities rather than the development utilities. The question only relates to the semantics of the options specified within the XPG; implementation specific extensions should not be documented.

Section 3.2 Development Utilities

3.2.1 Supported Commands

Question 3: *Which of the development utilities defined in the XPG are not provided with the implementation?*

Answer:

The sdb utility is not provided.

Rationale

The XPG Volume 1 states that “The development utilities might not be present in all X/Open compliant systems; in designated (**DEVELOPMENT**) systems all of the development utilities must be present and must conform to the published definition.”

Reference

XPG3 Volume 1, page 2 - *Status of Interfaces*

3.2.2 Command behavior

Question 4: *In what ways do the development utilities provided by the implementation behave differently from the specifications contained in the XPG?*

Answer:

The make utility looks for sccs s.files in the directory ./SCCS instead of in the current directory.

Rationale

This question provides a greater degree of granularity than the previous question, requiring the semantic differences associated with the development utilities to be specified. The question only relates to the semantics of the options specified within the XPG; implementation-specific extensions should not be documented.

Section 3.3 Internationalization Option

3.3.1 Commands and Utilities

Question 5: *Is an internationalized environment, reflecting changes in the locale setting as described in* **XPG Volume 1- XSI Commands and Utilities**, *supported*?

Answer:

Command	Behavior Specified in XPG3	Supported
ar	LC_TIME affects date format	Yes
awk	LC_COLLATE, LC_CTYPE affect regular expression matching	No

(continued)

Command	Behavior Specified in XPG3	Supported
	LC_COLLATE affects the behavior of string comparisons	No
	LC_NUMERIC affects the behavior of the radix character	No
comm	LC_COLLATE affects sorting sequence	No
cp,ln,mv	LANG affects yes string	Yes
cpio	LC_COLLATE, LC_CTYPE affect filename pattern matching	No
	LC_TIME affects date format	Yes
date	LC_TIME affects date formatting options	Yes
ed,red	LC_COLLATE, LC_CTYPE affects regular expression matching	No
	LC_CTYPE is used to determine whether characters are printable	Yes
egrep	LC_COLLATE, LC_CTYPE affects regular expression matching	No
	LC_CTYPE is used to determine character classification (alphabetic, upper case, lower case)	Yes
expr	LC_COLLATE, LC_CTYPE affects regular expression matching	No
	LC_COLLATE affects the behavior of relational operators	No
fgrep	LC_CTYPE is used to determine character classification (alphabetic, upper-case, lower case)	Yes
find	LANG affects yes string	No
	LC_COLLATE, LC_CTYPE affects filename pattern matching	No

(continued)

Command	behavior Specified in XPG3	Supported
grep	LC_COLLATE, LC_CTYPE affects regular expression matching	No
	LC_CTYPE is used to determine character classification (alphabetic, upper-case, lower case)	Yes
join	LC_COLLATE affects sorting sequence	No
lpstat	LC_TIME affects date format	Yes
ls	LC_COLLATE affects sorting sequence	Yes
	LC_CTYPE is used to determine whether a character is printable	Yes
	LC_TIME affects date format	Yes
mail	LC_TIME affects date format	Yes
mailx	LC_COLLATE, LC_CTYPE affects filename pattern matching	No
	LC_TIME affects date format	Yes
pg	LC_COLLATE, LC_CTYPE affects filename pattern matching	No
pr	LC_TIME affects date format	Yes
	LC_CTYPE is used to determine whether a character is printable	Yes
ps	LC_TIME affects date format	Yes
`rm,rmdir`	`LANG` affects yes string	No
`sed`	`LC_COLLATE, LC_CTYPE` affects regular expression matching	No
	`LC_CTYPE` is used to determine whether a character is printable	Yes
`sh`	`LC_COLLATE, LC_CTYPE` affects filename pattern matching	No

(continued)

Command	behavior Specified in XPG3	Supported
	LC_CTYPE is used to determine whether a character is alphabetic	No
sort	LC_COLLATE affects sorting sequence	No
	LC_CTYPE affects character classification (alphabetic, upper case, printing)	Yes
	LC_NUMERIC affects the determination of the radix character	No
tar	LC_TIME affects date format	No
	LANG affects yes string	No
tr	LC_COLLATE, LC_CTYPE affects bracketed expressions	No
	LC_CTYPE affects the definition of the character universe	No
uniq	LC_COLLATE affects sorting sequence	No
uucp	LC_TIME affects date format	No
uustat	LC_TIME affects date format	No
wc	LC_CTYPE is used to determine white-space characters	Yes
who	LC_TIME affects date format	Yes
yacc	LC_CTYPE is used to determine character classification	Yes

Rationale

This behavior is collectively optional, that is, it should be provided for all commands listed (subject to sections 3.1 and 3.2, which identify those commands not supplied by the vendor and those which do not fully support the X/Open specification).

Reference

XPG3 Volume 1, pages 4-5 - *Status of Interfaces.*

3.3.2 Regular Expressions in Commands

Question 6: *Which form of regular expression syntax is supported by those commands which use regular expressions?*

Answer:

Command	Regular Expression Syntax Supported
awk	Extended
csplit	Simple
ed	Simple
egrep	Extended
ex	Simple
expr	Simple
grep	Simple
lex	Extended
pg	Simple
sdb	sdb is not supported
sed	Simple
vi	Simple

Rationale

The XPG Volume 3 - *XSI Supplementary Definitions* requires that an internationalized set of commands will provide regular expression syntax for the above commands in one of the forms specified for that command. The XPG encourages the implementation of internationalized regular expressions for all of the above utilities. It should be noted that the `sdb` command is an optional development utility and may not be available on all XPG conforming systems.

Reference

XPG3 Volume 3, pages 49-51 - *Regular Expressions*

Chapter 4: C Language

SPARC

Product Identification

Product Identification	SPARCompiler C
Version / Release No.	2.0.1

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

VSX Test Suite Release	VSX 4.3.2
Testing Agency Name	Mindcraft, Inc.
Address	410 Cambridge Avenue Palo Alto, California 94306

x86

Product Identification

Product Identification	ProCompiler C
Version / Release No.	2.0.1

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

VSX Test Suite Release	VSX 4.3.2
Testing Agency Name	Mindcraft, Inc.
Address	410 Cambridge Avenue Palo Alto, California 94306

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC and x86, running Solaris. Installation procedures are provided in *SPARC: Installing Solaris Software or x86: Installing Solaris Software.*

Temporary Waivers

List below references to any temporary waivers granted by X/Open in respect to minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

Section 4.1 Implementation Limits

Question 1: *What limits does the implementation impose on the significant part of an identifier?*

Answer:

External identifiers	No limits; all characters are significant
Non-External identifiers	No limits; all characters are significant

Rationale

The XPG states that, while there is no limit to the length of an identifier, only a certain number of characters are significant. The XPG points out that there must be at least eight characters for a non-external name, but may be less for external names.

Reference

XPG 3 Volume 4, page 3 - *Lexical Conventions*

4.2 General

Question 2: *What truncation rules are applied when a floating value is converted to an integral value?*

Answer:

Truncation of floating point values is always towards zero.

Rationale

The XPG states that such conversions are machine dependent. In particular, the XPG points out the differences related to the truncation of negative numbers.

Reference

XPG Volume 4, page 10 - *Conversions*

Question 3: *What truncation rules are applied when using the division operator and either of the operands is negative?*

Answer:

Truncation towards zero

Rationale:

The XPG states that such truncations are machine dependent.

Reference

XPG Volume 4, page 16 - *Expressions*

Chapter 11: Terminal Interfaces

Product Identification

Product Identification	Solaris
Version / Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

None

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC and x86, running Solaris. Installation procedures are given in *SPARC: Installing Solaris Software or x86: Installing Solaris Software*.

Temporary Waivers

List below references to any temporary waivers granted by X/Open in respect to minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

Chapter 12: Window Management

Product Identification

Product Identification	OpenWindows
Version/Release No.	3.0 and subsequent releases

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

None

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC and x86, running Solaris. Installation procedures are provided in *SPARC: Installing Solaris Software or x86: Installing Solaris Software*.

Temporary Waivers

List below references to any temporary waivers granted by X/Open in respect to minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

Chapter 14: Inter-process Communication

Product Identification

Product Identification	Solaris
Version/Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

None

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC and x86, running Solaris. Installation procedures are provided in *SPARC: Installing Solaris Software or x86: Installing Solaris Software*.

Temporary Waivers

List below references to any temporary waivers granted by X/Open in respect to minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

Chapter 15: Source Code Transfer

Section 15.1 Utilities

Product Identification

Product Identification	Solaris
Version/Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance Reference

Indicator of Compliance

None

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC and x86, running Solaris. Installation procedures are provided in *SPARC: Installing Solaris Software or x86: Installing Solaris Software*.

SPARC

For floppy disk hardware:

SPARCstation with external or internal floppy disk drive (part number X554H)

For magnetic tape hardware:

SPARC Office Servers (SPARCsystem 330 or SPARCsystem 470) with 1/2-inch tape drive subsystem (part number 680A), and
SPARC Data Center Servers (SPARCserver 390 or SPARCserver 490) with 1/2-inch tape drive subsystem (part numbers 682A or 683A)

x86

For Source Code Transfer software:

x86, running Solaris 2.5. Installation procedures are given in the *Solaris System Configuration and Installation Guide.*

For floppy disk hardware:

x86 machine, with external or internal floppy disk drive

Temporary Waivers

List below references to any temporary waivers granted by X/Open regarding minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

Formats

Question 1: *Which exchange media format(s) may be written by the system?*

Answer:

80 track floppy disk	Yes
40 track floppy disk	No
1600 bpi PE magnetic tape	Yes

Rationale

XPG3 states that standards are referenced for transfer of floppy disks and magnetic tapes between machines. Because of the different nature of X/Open conformant systems, it is not possible to define a single portable medium which is supported across the whole range of systems.

Reference

XPG3 Volume 3, Chapters 15, 16, and 17

Question 2: *Which exchange media format(s) may be read by the system?*

80 track floppy disk	Yes
40 track floppy disk	No
1600bpi PE magnetic tape	Yes

Rationale

XPG3 states that standards are referenced for transfer of floppy disks and magnetic tapes between machines. Because of the different nature of X/Open conformant systems, it is not possible to define a single portable medium which is supported across the whole range of systems. In addition, some systems can read a wider range of formats than they can write.

Reference

XPG3 Volume 3, Chapters 15, 16, and 17

Utilities

Question 3: *Which utilities are used to create and read the archive formats specified in* **XPG Volume 3-XSI Supplementary Definitions***?*

Answer:

Format	**Creating**	**Reading**
Extended `tar`	`cpio -H USTAR`	`cpio -H USTAR`
`cpio`	`cpio -H odc`	`cpio -H odc`

Rationale

There is no explicit definition as to the commands that must be used to create and retrieve these archives. On most systems this will be achieved by the `tar` and `cpio` commands. There are other commands available which produce these archives. On some implementations the command may need a special option to enable reading of the specified formats with the "standard" option being to create archives which are backwards compatible with previous versions of the command.

Reference

XPG3 Volume 3, page 151-2 – Utilities

Invalid Files Names

Question 4: *What file name is used to contain data from the archive in the case that the file name on the archive is invalid for the system on which the file hierarchy is being created?*

Answer:

Format	**File Name**
Extended `tar`	All legal file names in a USTAR archive are legal in the filesystem.
`cpio`	All legal file names in a cpio archive are legal in the filesystem.

Rationale

Because an archive can contain non-portable file names it is necessary for an archive reading utility to be able to generate a file and store the data associated with a non-portable file name when this is encountered on the archive. There may be a need to generate a number of such file names in the same directory and the specification should detail the algorithm used to generate these file names.

Reference

XPG3 Volume 3, page 151– Utilities

Multivolume Archives

Question 5: *How does the archive reading utility determine which file to read as the next volume when an end-of-media condition is encountered?*

Answer:

Format	**Method**
Extended `tar`	The `tar` utility prompts the user for the pathname of the next file in the archive. (The path need not name a device.)
`Cpio`	The `cpio` utility prompts the user for the pathname of the next file in the archive. (The path need not name a device.)

Rationale

In many cases the utility will prompt the user for the path name of the device to use for the next volume. There may be extensions to the utility syntax that allow the definition of alternate addresses for subsequent volumes.

Reference

XPG3 Volume 3, page 151-2 – Utilities.

X/Open Specification and Related Publications

The X/Open Portability Guide is published by Prentice-Hall in the United States. The set is comprised of the seven volumes listed below; the ISBN number follows the volume title:

- *Volume 1: XSI Commands and Utilities, 0-13-68555835-X*
- *Volume 2: XSI System Interface and Headers, 0-13-685843-0*

- *Volume 3: XSI Supplementary Definitions, 0-13-685850-3*
- *Volume 4: Programming Languages, 0-13-685868-6*
- *Volume 5: Data Management, 0-13-685876-7*
- *Volume 6: Window Management, 0-13-685884-8*
- *Volume 7: Networking Services, 0-13-685892-9*

X/Open and XPG4 6

This chapter discusses the compliance of Solaris 2.5 to the programming interface specifications contained in the X/Open Portability Guide Issue 4, (XPG4).

The X/Open Portability Guide, Issue 4

In 1992, X/Open published the X/Open Portability Guide, Issue 4. XPG4 retains compliance to the POSIX.1–1988 standard but is extended to the ISO/IEC updated POSIX.1 standard and the ISO/IEC C language standard. The X/Open Portability Guide Issue 3 remains available and system and components can still be branded to XPG3, but XPG4 branding offers significant additional capability.

This chapter identifies Solaris 2.5 as a conforming implementation of XPG4, and displays the XPG4 Base brand trademark. It also presents the X/Open Conformance Statement, which documents Solaris' compliance to the programming interface specifications of the X/Open Portability Guide, Issue 4.

The X/Open Brand Trademark

X/Open provides a verification and branding program that developers can use to show that their products are X/Open compliant. Sun Microsystems has been a strong supporter of the X/Open branding process since its inception.

Figure 6-1 The XPG4 Base Brand Logo

X/Open
XPG4
BASE PROFILE

Solaris is available as an XPG4 and/or XPG3 BASE branded configuration. Solaris products and software products from independent software vendors that have received XPG4 branding are described below:

- **Window Management (OpenWindows)**—The Solaris window system, which supports the OPEN LOOK Graphical User Interface, has earned the XPG4 brand by implementing the programmer's interface to the X Window System. OpenWindows supports the Window Management component of the X/Open BASE level.
- **Internationalized System Calls and Libraries**— Solaris meets the Internationalized System Calls and Libraries component requirements of the BASE level.
- **Commands and Utilities**—X/Open's specification of standard interfaces for utilities allows for portable shell scripts. Solaris meets the Commands and Utilities component requirements of the BASE system.
- **ProCompiler™ C 2.0.1**— The ProCompiler for Solaris for x86 is fully conformant with the ANSI/ISO Standard for C. It has passed X/Open verification test suite VSX 4.3.2 and meets the C Language component requirements of the BASE level.
- **SPARCompiler C 2.0.1**—The SPARCompiler for the C programming language based on Common Usage C has passed X/Open verification test suite VSX 4.3.2 and meets the C Language component requirements of the BASE level.
- **Sun FORTRAN 3.0**—Sun's compiler for the FORTRAN programming language is fully compliant with the definition in the American National Standards Institute (ANSI) document.
- **Sun Pascal 3.0.1**—Sun's compiler for the Pascal programming language is fully compliant with the ISO standard.

- **Magnetic Media (Source Code Transfer)**—Sun conforms to X/Open's specifications for transferring source code between machines with compatible media and facilitating the transfer of source code in machine-readable form. Solaris supports the Source Code Transfer component of the X/Open BASE level.
- **Inter-Process Communication**—Sun supports X/Open's specifications for interfaces providing message queue, semaphore, and shared memory facilities for communication and synchronization between processes. The SunOS operating system fulfills the requirements of the Inter-Process Communications component of the X/Open BASE level.
- **Terminal Interfaces (XSI Curses Interface)**—The XSI Curses Interface meets X/Open's specifications for providing a generic terminal interface that is independent of terminal hardware or connection methods for updating screens on character-oriented and block-oriented terminals. Solaris systems fulfill the requirements of the Terminal Interfaces component of the X/Open BASE level.

The X/Open Conformance Statement for Solaris

The remaining pages of this chapter feature the X/Open Conformance Statement for Solaris.

X/Open Conformance Statement

X/OPEN Conformance Statement Questionnaire

1. Internationalized System Calls and Libraries

Product Identification

Product Identification	Solaris
Version/Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

Indicator of Compliance

VSX Test Suite Release	VSX 4.3.4
Testing Agency Name	Mindcraft, Inc.
Address	410 Cambridge Avenue Palo Alto, California 94306

Environment Specification

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC

SPARC running Solaris 2.5. Installation procedures are provided in SPARC: Installing Solaris Software

To reproduce the test environment, do these steps:

1. Edit the `/etc/saf/zsmon/_pmtab` file to turn off the ttysoftcarrier detect:

   Change the `ttya` and `ttyb` fields from `:y:` to `:n:`. (The colons (:) act as field separators.)

2. Verify that the ttymodes settings in the `/kernel/drv/options.conf` file are set to:

```
2502:1805:bd:8a3b:3:1c:7f:15:4:0:0:0:11:13:1a:19:12:f:17:16
```

3. Disable `ypbind` to allow rebooting of the system:

   a. `cd /usr/lib/netsvc/yp`

   b. `mv ypbind ypbind-`

4. Set the `eeprom` variables that affect the `tty`:

   a. On the keyboard, hit `STOP-A` to display the `prom` prompt.

   b. At the prompt, execute the following steps:

```
setenv ttya-ignore-cd false
setenv ttyb-ignore-cd false
setenv ttya-rts-dtr-off false
setenv ttyb-rts-dtr-off false
```

5. Reboot the system

Note – When installing Solaris, set the time zone by selecting a time zone format that conforms to the POSIX.1 format for TZ defined on page 152 and page 153 of the IEEE Std. 1003.1–1990.

Note – The following option must be added to any compiler line: `-D_XOPEN_VERSION=4`

Note – The following must be added to any link/load line: `/usr/ccs/lib/values_xpg4.o`

---

x86

x86 running Solaris 2.5. Installation procedures are provided in x86: Installing Solaris Software.

To reproduce the test environment, follow these steps:

1. Become `root`.

2. Ensure the correct serial port links:

- `/dev/ttya` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/term/a` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/tty00` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/ttyb` should be a link to `/devices/isa/asy@2f8,0:a`
- `/dev/term/b` should be a link to `/devices/isa/asy@2f8,0:a`
- `/dev/tty01` should be a link to `/devices/isa/asy@2f8,0:b`

    a. If the `/dev/tty01` link is missing, perform the following:

- Edit `/kernel/drv/asy.conf` and uncomment the COM2 entry
- `# touch /reconfigure`

3. Set the correct serial port permissions:

- `# chmod 666 /devices/eisa/asy*`

4. Turn off the `ttysoftcarrier` detect:
   Using an editor such as `vi`, in the `/etc/saf/zsmon/_pmtab` file, change the next to last field for both the `ttya` entry and the `ttyb` entry from `y` to `n` (the colon (`:`) acts as the field separator):

- `# vi /etc/saf/zsmon/_pmtab`

5. Reboot the system.

Note – When installing Solaris, set the time zone by selecting a time zone format that conforms to the POSIX.1 format for TZ defined on page 152 and page 153 of the IEEE Std. 1003.1–1990.

Note – The following option must be added to any compiler line:
-D_XOPEN_VERSION=4

Note – The following must be added to any link/load line:
/usr/ccs/lib/values_xpg4.o

Temporary Waivers

None.

Section 1.1: General Attributes

1.1.1 XPG4 Feature Groups

Question 1: *Which of the following Feature Groups are supported by the implementation?*

Response:

POSIX C–language Binding	Yes
Shared Memory	Yes
Encryption	Yes
Enhanced Internationalization	Yes

Note: All the interfaces in all these groups must exist on all XPG4 XSI-conformant systems, and each interface must either behave according to the description in System Interfaces and Headers, Issue 4, or indicate an error, with *errno* set to [ENOSYS].

Support for particular Feature Groups may be required in order to conform to particular X/Open Profiles. Support for a Feature Group can only be claimed if *all* interfaces in that group behave according to the relevant descriptions in System Interfaces and Headers, Issue 4.

Rationale

System Interfaces and Headers, Issue 4 states that the system may provide one or more of the Feature Groups listed.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Section 1.2, Conformance and Section 1.3, *Feature Groups*.

1.1.2 POSIX.1 Supported Features

Question 2 *Which of the following options, specified in the* `<unistd.h>` *header file, are available on the system?*

Response:

Macro Name	Meaning	Provided
_POSIX_CHOWN_RESTRICTED	The use of chown() is restricted to a process with appropriate privileges and to changing the group ID of a file only to the effective group ID of the process or one of its supplementary groups IDs.	Yes
_POSIX_NO_TRUNC	Pathname components longer than {NAME_MAX} generate an error	Yes

(continued)

Macro Name	Meaning	Provided
_POSIX_VDISABLE	Terminal special characters defined in `<termios.h>` can be disabled using this character value.	Yes
_POSIX_SAVED_IDS	Each process has a saved set-user-ID and saved set-group-ID.	Yes
_POSIX_JOB_CONTROL	Implementation supports job control.	Yes

Rationale

For an XSI–conformant implementation, all of these POSIX features must be provided. In some cases the feature need not be provided for all files or devices supported by the implementation.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 4, Headers `<unistd.h>`.

1.1.3 Float, Stdio and Limit Values

Question 3: *What are the values associated with the following constants specified in the* **<float.h>** *header file?*

Response:

Macro Name	Meaning	Value
FLT_RADIX	Radix of the exponent representation.	2
FLT_MANT_DIG	Number of base-FLT_RADIX digits in the float significand.	24
DBL_MANT_DIG	Number of base-FLT_RADIX digits in the double significand.	53
LDBL_MANT_DIG	Number of base_FLT_RADIX digits in the long double significand.	64

(continued)

Macro Name	Meaning	Value
FLT_DIG	Number of decimal digits, *q*, such that any floating point number with *q* digits can be rounded into a float representation and back again without change to the *q* digits.	6
DBL_DIG	Number of decimal digits, *q*, such that any floating point number with *q* digits can be rounded into a double representation and back again without change to the *q* digits.	15
LDBL_DIG	Number of decimal digits, *q*, such that any floating point number with *q* digits can be rounded into a long double representation and back again without change to the *q* digits.	18
FLT_MIN_EXP	Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized float.	-125
DBL_MIN_EXP	Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized double.	-1024
LDBL_MIN_EXP	Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized long double.	-16381
FLT_MIN_10_EXP	Minimum negative integer such that 10 raised to that power is in the range of normalized floats.	-125
DBL_MIN_10_EXP	Minimum negative integer such that 10 raised to that power is in the range of normalized doubles.	-307
LDBL_MIN_10_EXP	Minimum negative integer such that 10 raised to that power is in the range of normalized long doubles.	-4931

(continued)

Macro Name	Meaning	Value
FLT_MAX_EXP	Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite float.	128
DBL_MAX_EXP	Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite double.	1024
LDBL_MAX_EXP	Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite long double.	16384
FLT_MAX_10_EXP	Maximum integer such that 10 raised to that power is in the range of representable finite floats.	38
DBL_MAX_10_EXP	Maximum integer such that 10 raised to that power is in the range of representable finite doubles.	308
LDBL_MAX_10_EXP	Maximum integer such that 10 raised to that power is in the range of representable finite long doubles.	4932
FLT_MAX	Maximum representable finite float.	3.402823466E+38F
DBL_MAX	Maximum representable finite double.	1.7976931348623157E+308
LDBL_MAX	Maximum representable finite long double.	1.189731495357231765085759326628007016E+4932L
FLT_EPSILON	Difference between 1.0 and the least value greater than 1.0 that is representable as a float.	1.192092896E-07F
DBL_EPSILON	Difference between 1.0 and the least value greater than 1.0 that is representable as a double.	2.2204460492503131E-16

(continued)

Macro Name	Meaning	Value
LDBL_EPSILON	Difference between 1.0 and the least value greater than 1.0 that is representable as a long double.	1.9259299443872358530559779425849273I9E-34L
FLT_MIN	Minimum normalized positive float.	1.175494351E-38F
DBL_MIN	Minimum normalized positive double.	2.2250738585072014E-308
LDBL_MIN	Minimum normalized positive long double.	3.3621031431120935062626778173217522603E-4932L

Rationale

This set of constants provides useful information regarding the underlying architecture of the implementation.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 4, *Headers*, <**float.h**>.

Question 4 *What are the values associated with the following constants (optionally specified in the* **<limits.h>** *header file)?*

Response:

Macro Name	Meaning	Minimum	Maximum
ARG_MAX	Maximum length of argument to the `exec` functions including the environment data.	4096	1048320
CHILD_MAX	Maximum number of processes per user ID.	6	See note
LINK_MAX	Maximum number of links to a single file.	8	32767
MAX_CANON	Maximum number of bytes in a terminal canonical input line.	255	256

(continued)

Macro Name	Meaning	Minimum	Maximum
NAME_MAX	Maximum number of bytes allowed in a terminal input queue.	14	See note
OPEN_MAX	Maximum number of open files that one process can have open at any one time.	16	See note
PATH_MAX	Maximum number of bytes in a pathname (including the terminating null).	255	See note
PIPE_BUF	Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.	512	5120
STREAM_MAX	Number of streams one process can have open at one time.	8	8
TZ_NAME_MAX	Maximum number of bytes supported for the name of a time zone.	3	3

Notes:
CHILD_MAX depends on how the system kernel is configured.

The maximum values for NAME_MAX and PATH_MAX vary depending on the file system type, but always provide at least the minimum requirement. The most common values are 255 for NAME_MAX and 1024 for PATH_MAX. Values for a specific path are available using pathconf().

OPEN_MAX defaults to 64, but users can increase or decrease this value using routines not specified by POSIX.1 or XPG4.

Rationale

Each of these limits can vary within bounds set by the Systems Interfaces and Headers, Issue 4. The minimum value that a limit can take on any XSI conforming system is given in the corresponding _POSIX_ value. A specific conforming implementation may provide a higher minimum value than this and the maximum value that it provides can differ from the minimum. Some conforming implementations may provide a potentially infinite value as the maximum, in which case the value is considered to be indeterminate. The minimum value must always be definitive since the _POSIX_ value provides a known lower bound for the range of possible values.

Reference
X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 4, *Headers*, <limits.h>

Question 5: *What are the values associated with the following constants specified in the* **<limits.h>** *header file?*

Response:

Macro Name	Meaning	Min.	Maximum
BC_BASE_MAX	Maximum *ibase* and *obase* values allowed by the `bc` utility.	99	32767 (`SHRT_MAX`)
BC_DIM_MAX	Maximum number of elements permitted in an array by the `bc` utility.	2048	32768 (`SHRT_MAX+1`)
BC_SCALE_MAX	Maximum scale value allowed by the `bc` utility.	99	32767 (`SHRT_MAX`)
BC_STRING_MAX	Maximum length of a string constant accepted by the `bc` utility.	1000	2048 (`LINE_MAX`)
COLL_WEIGHTS_MAX	Maximum number of weights that can be assigned to an entry of the LC_COLLATE order keyword in the local definition file.	2	4
EXPR_NEST_MAX	Maximum number of expressions that can be nested within parentheses by the `expr` utility.	32	32
LINE_MAX	Maximum length in bytes including the trailing newline of a utility's input line when the utility is described as processing text files.	2048	2048

(continued)

Macro Name	Meaning	Min.	Maximum
NGROUPS_MAX	Maximum number of simultaneous supplementary group IDs per process.	16	16
RE_DUP_MAX	Maximum number of repeated occurrences of a regular expression permitted when using interval notation.	255	255

Rationale

Each of these limits can vary within bounds set by the System Interfaces and Headers, Issue 4. The minimum value that a limit can take on any XSI conforming system is given in the corresponding _POSIX_ or POSIX2_ value. A specific conforming implementation may provide a higher minimum value than this and the maximum value that it provides can differ from the minimum. Some conforming implementations may provide a potentially infinite value as the maximum, in which case the value is considered to be indeterminate. The minimum value must always be definitive since the _POSIX_ or _POSIX2_ value provides a known lower bound for the range of possible values.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 4, *Headers*, <`limits.h`>

Question 6:*What are the values associated with the following numerical constants specified in the* <`limits.h`> *header file?*

Response:

Macro Name	Meaning	Value
CHAR_BIT	Number of bits in a char.	8
CHAR_MAX	Maximum value of a char.	127
INT_MAX	Maximum value of an int.	2147483647

(continued)

Macro Name	**Meaning**	**Value**
LONG_BIT	Number of bits in a long int.	32
LONG_MAX	Maximum value of a long int.	2147483647L
MB_LEN_MAX	Maximum number of bytes in a character, for any supported locale.	5
SCHAR_MAX	Maximum value of a signed char.	127
SHRT_MAX	Maximum value of a short.	32767
SSIZE_MAX	Maximum value of an object of type ssize_t.	2147483647
UCHAR_MAX	Maximum value of an unsigned char.	25
UINT_MAX	Maximum value of an unsigned int.	4294967295U
ULONG_MAX	Maximum value of an unsigned long int.	4294967295UL
USHRT_MAX	Maximum value of an unsigned short int.	65535
WORD_BIT	Number of bits in a word or int.	32

Rationale

This set of constants provides useful information regarding the underlying architecture of the implementation.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 4, *Headers*, **<limits.h>** .

Question 7: *What are the values associated with the following numerical constants specified in the* **<stdio.h>** *header file?*

Response:

Macro Name	Meaning	Value
FOPEN_MAX	Number of streams which the implementation guarantees can be open simultaneously.	20 (SPARC) 60 (x86)
L_tmpnam	Maximum size of character array to hold tmpnam() output.	25
TMP_MAX	Minimum number of unique filenames generated by tmpnam(), which is the maximum number of times an application can call tmpnam() reliably.	17567

Rationale

This set of constants provides useful information about the implementation.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 4, *Headers*, **<stdio.h>**

1.1.4 Error Conditions

Question 8: *Which of the following option errors listed in the System Interfaces and Headers, Issue 4 are detected in the circumstances specified?*

Response:

Function	Error	Detected
access()	EINVAL	Yes
	ETXTBSY	No
acos()	EDOM	Yes
asin()	EDOM	Yes
	ERANGE	No
	ERANGE	No

(continued)

Function	Error	Detected
atan()	EDOM	Yes
atan2()	EDOM	Yes
	ERANGE	Yes
catclose()	EBADF	No
	EINTR	No
catgets()	EBADF	No
	EINTR	No
catopen()	EACCES	Yes
	EMFILE	Yes
	ENAMETOOLONG	Yes
	ENFILE	Yes
	ENOENT	Yes
	ENOMEM	Yes
	ENOTDIR	Yes
ceil()	EDOM	Yes
cfsetispeed()	EINVAL	No
cfsetospeed()	EINVAL	No
chmod()	EINVAL	No
chown()	EINVAL	Yes
closedir()	EBADF	Yes
	EINTR	No
cos()	EDOM	Yes
	ERANGE	No
cosh()	EDOM	Yes
erf()	EDOM	Yes
	ERANGE	No
erfc()	EDOM	Yes
	ERANGE	No
exec()	ENOMEM	Yes
	ETXTBSY	No

(continued)

Function	Error	Detected
exp()	EDOM	No
	ERANGE	Yes
fabs()	EDOM	Yes
	ERANGE	No
fcntl()	EDEADLK	Yes
fdopen()	EBADF	No
	EINVAL	No
	EMFILE	Yes
	ENOMEM	Yes
fgetc()	ENOMEM	Yes
	ENXIO	Yes
fgetwc()	ENOMEM	Yes
	ENXIO	Yes
	EILSEQ	Yes
fileno()	EBADF	No
floor()	EDOM	Yes
fmod()	EDOM	Yes
	ERANGE	No
fopen()	EINVAL	No
	EMFILE	Yes
	ENOMEN	Yes
	ETXTBSY	No
fork()	ENOMEM	Yes
fpathconf()	EBADF	Yes
	EINVAL	Yes
fprintf()	EINVAL	Yes
	EILSEQ	Yes
fputc()	ENOMEM	Yes
	ENXIO	Yes

(continued)

Function	Error	Detected
fputwc()	ENOMEM	Yes
	ENXIO	Yes
	EILSEQ	Yes
freopen()	EINVAL	No
	ENOMEM	No
	ETXTBSY	No
frexp()	EDOM	Yes
fseek()	EINVAL	Yes
	EPIPE	No
ftw()	EINVAL	No
getcwd()	EACCES	Yes
	ENOMEM	Yes
getgrgid()	EIO	Yes
	EINTR	No
	EMFILE	Yes
	ENFILE	Yes
getgrnam()	EIO	Yes
	EINTR	No
	EMFILE	Yes
	ENFILE	Yes
getlogin()	EMFILE	Yes
	ENFILE	Yes
	ENXIO	Yes
getpass()	EINTR	No
	EIO	No
	EMFILE	Yes
	ENFILE	No
	ENXIO	No
getpwnam()	EIO	Yes
	EINTR	No
	EMFILE	Yes
	ENFILE	Yes

(continued)

Function	Error	Detected
getpwuid()	EIO	Yes
	EINTR	Yes
	EMFILE	Yes
hsearch()	ENOMEM	Yes
hypot()	EDOM	Yes
	ERANGE	No
iconv()	EBADF	Yes
iconv_close()	EBADF	Yes
iconv_open()	EMFILE	Yes
	ENFILE	Yes
	ENOMEM	Yes
	EINVAL	Yes
isatty()	EBADF	No
	ENOTTY	No
j0()	EDOM	No
	ERANGE	Yes
j1()	EDOM	No
	ERANGE	Yes
jn()	EDOM	No
	ERANGE	Yes
ldexp()	EDOM	Yes
	ERANGE	No
lgamma()	EDOM	No
	ERANGE	Yes
link()	EPERM	Yes
	EXDEV	Yes
log()	EDOM	Yes
	ERANGE	No
log10()	EDOM	Yes
	ERANGE	No

(continued)

Function	Error	Detected
mblen()	EILSEQ	Yes
mbstowcs()	EILSEQ	Yes
mbtowc()	EILSEQ	Yes
modf()	EDOM	Yes
	ERANGE	No
open()	EINVAL	No
	ETXTBSY	No
opendir()	EMFILE	Yes
	ENFILE	Yes
pathconf()	EACCES	Yes
	EINVAL	Yes
	ENAMETOOLONG	Yes
	ENOENT	Yes
	ENOTDIR	Yes
pow()	EDOM	Yes
	ERANGE	No
putenv()	ENOMEM	Yes
read()	ENXIO	Yes
readdir()	EBADF	Yes
rename()	ETXTBSY	No
setvbuf()	EBADF	No
sigaction()	EINVAL	Yes
sigaddset()	EINVAL	Yes
sigdelset()	EINVAL	Yes
sigismember()	EINVAL	Yes
signal()	EINVAL	Yes
sin()	EDOM	No
	ERANGE	Yes

(continued)

Function	Error	Detected
sinh()	EDOM	No
	ERANGE	Yes
sqrt()	EDOM	Yes
strcoll()	EINVAL	No
strerror()	EINVAL	No
strxfrm()	EINVAL	No
tan()	EDOM	Yes
	ERANGE	No
tanh()	EDOM	Yes
	ERANGE	No
tcdrain()	EIO	Yes
tcflush()	EIO	Yes
tcsendbreak()	EIO	Yes
tcsetattr()	EIO	Yes
tmpfile()	EMFILE	Yes
	ENOMEM	Yes
ttyname()	EBADF	Yes
	ENOTTY	Yes
ungetwc()	EILSEQ	Yes
unlink()	ETXTBSY	No
wcscoll()	EINVAL	Yes
wcstombs()	EILSEQ	Yes
wcsxfrm()	EINVAL	Yes
write()	ENXIO	Yes
y0()	EDOM	Yes
	ERANGE	No
y1()	EDOM	Yes
	ERANGE	No

(continued)

Function	Error	Detected
yn()	EDOM	Yes
	ERANGE	No

Rationale

Each of the above error conditions is marked as optional in System Interfaces and Headers, Issue 4 and an implementation may return this error in the circumstances specified or may not provide the error indication.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Section 2.3, *Error Numbers*.

1.1.5 Mathematical Interfaces

Question 9: *What format of floating-point numbers is supported by this implementation?*

Response:

IEEE floating point format.

Rationale

Most implementations support IEEE floating point format either in hardware or software. Some implementations support other formats with different exponent and mantissa accuracy. These differences need to be defined.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Section 1.6, *Relationship to Formal Standards*.

1.1.6 Data Encryption

Question 10: *Are the optional data encryption interfaces provided?*

Response:

Function	Provided
crypt()	Yes

(continued)

Function	Error	Detected
encrypt()		Yes (Decryption capabilities not provided to areas restricted by U.S. export law.)
setkey()		Yes

Rationale

Normally, an implementation will either provide all three of these routines or will provide none of them at all. If the routines are not provided, then the implementation must provide a dummy interface which always raises an ENOSYS error condition.

It is also possible that the implementation of the *encrypt()* function may be affected by export restrictions, in which case, the restrictions should be documented here.

For example, historical implementations have supplied all three of these routines outside the U.S.A., but due to export restrictions on the decoding algorithm, a dummy version of *encrypt()* is provided that does encoding but no decoding.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Section 1.2, *Conformance*.

Section 1.2 Process Handling

1.2.1 Process Generation

Question 11: *Which file types (regular, directory, FIFO, special, and so on) are considered to be executable?*

Response:

Only regular file types may be executed.

Rationale

The [EACCES] error associated with `exec` functions occurs in circumstances when the implementation does not support execution of files of the type specified. A list of these file types needs to be provided.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 3, *Functions*, `exec`.

Section 1.3 File Handling

1.3.1 Access Control

Question 12: *What file access control mechanisms does the implementation provide?*

Response:

There is no additional or optional file access control mechanism.

Rationale

System Interfaces and Headers, Issue 4 notes that implementations may provide *additional* or *alternate* file access control mechanisms, or both.

Reference

X/Open CAE Specification, System Interfaces Definitions, Issue 4, Chapter 2, *Glossary,* file access permissions.

1.3.2 Files and Directories

Question 13: *Are any additional or alternate file access control mechanisms implemented that could cause fstat() or stat() to fail?*

Response:

There is no additional or optional file access control mechanism.

Rationale

System Interfaces and Headers, Issue 4 notes that there could be an interaction between additional and alternate access controls and the success of `fstat()` and `stat()`. This would suggest that an implementation can allow access to a file but not allow the process to gain information about the status of the file.

Reference

X/Open CAE Specification, System Interfaces Definitions, Issue 4, Chapter 3, *Functions*, *fstat()* and *stat()*.

1.3.3 Formatting Interfaces

Question 14: *Does the* printf() *function produce character string representations for Infinity and NaN to represent the respective values?*

Response: Yes

Rationale

This behavior is often provided on systems with mathematical functions that produce these results.

Reference

X/Open CAE Specification, System Interfaces and Headers, Issue 4, Chapter 3, *Functions*, fprintf()**.**

Section 1.4: Internationalized System Interfaces

1.4.1 Coded Character Sets

Question 15: *What coded character sets are supported by the implementation?*

Response:

Solaris 2.4 supports the following coded character sets:

ASCII
ISO 8859-1
JIS X0201
JIS X0208
KS C 5601-87
GB 2312-80
CNS 11643-1986

Rationale

System Interface Definitions, Issue 4 states that conforming implementations support one or more coded character sets, and that each of these includes the portable character set.

Reference

X/Open CAE Specification, System Interfaces Definitions, Issue 4, Chapter 4, *Character Set*.

Question 16: *What is the implementation's underlying internal codeset?*

Response:

ISO 8859-1:1987

Rationale

It is useful to be aware of the underlying codeset of the implementation.

Reference

X/Open CAE Specification, System Interfaces Definitions, Issue 4, Chapter 4, *Character Set*.

1. Commands and Utilities

PRODUCT IDENTIFICATION

Product Identification	Solaris
Version/Release No.	2.5

If you do not supply this component yourself, please identify below the supplier you reference.

INDICATOR OF COMPLIANCE

VSX Test Suite Release	VSC 4.1.4
Testing Agency Name	SunSoft, A Sun Microsystems, Inc. Business
Address	2550 Garcia Avenue Mountain View CA 94043

ENVIRONMENT SPECIFICATION

Enter below details of the hardware and software environment in which testing took place, including compilation routines and installation procedures (if any). Sufficient detail must be supplied to enable conformant behavior and any test results to be reproduced.

SPARC

SPARC running Solaris 2.5. Installation procedures are provided in SPARC: Installing Solaris Software.

To reproduce the test environment, follow these steps:

1. Edit the `/etc/saf/zsmon/_pmtab` file to turn off the ttysoftcarrier detect:

Change the `ttya` and `ttyb` fields from `:y:` to `:n:`. (The colons (:) act as field separators.)

2. Verify that the ttymodes settings in the `/kernel/drv/options.conf` file are set to:

```
2502:1805:bd:8a3b:3:1c:7f:15:4:0:0:0:11:13:la:19:12:f:17:16
```

3. Disable `ypbind` to allow rebooting of the system:
   a. `cd /usr/lib/netsvc/yp`
   b. `mv ypbind ypbind-`
4. Set the `eeprom` variables that affect the `tty`:
   a. On the keyboard, hit `STOP-A` to display the `prom` prompt.
   b. At the prompt, execute the following steps:

```
setenv ttya-ignore-cd false
setenv ttyb-ignore-cd false
setenv ttya-rts-dtr-off false
setenv ttyb-rts-dtr-off false
```

5. Reboot the system

Note – When installing Solaris, set the time zone by selecting a time zone format that conforms to the POSIX.1 format for TZ defined on page 152 and page 153 of the IEEE Std. 1003.1–1990.

Note – The following option must be added to any compiler line: -D_XOPEN_VERSION=4

Note – The following must be added to any link/load line: /usr/ccs/lib/values_xpg4.o

---

x86

x86 running Solaris 2.5. Installation procedures are provided in x86: Installing Solaris Software.

To reproduce the test environment, follow these steps:

1. Become `root`.

2. Ensure the correct serial port links:

- `/dev/ttya` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/term/a` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/tty00` should be a link to `/devices/isa/asy@3f8,0:a`
- `/dev/ttyb` should be a link to `/devices/isa/asy@2f8,0:a`
- `/dev/term/b` should be a link to `/devices/isa/asy@2f8,0:a`
- `/dev/tty01` should be a link to `/devices/isa/asy@2f8,0:b`

   a. If the `/dev/tty01` link is missing, perform the following:

- Edit /`kernel/drv/asy.conf` and uncomment the COM2 entry
- `# touch /reconfigure`

3. Set the correct serial port permissions:

- `# chmod 666 /devices/eisa/asy*`

4. Turn off the `ttysoftcarrier` detect:
   Using an editor such as `vi`, in the `/etc/saf/zsmon/_pmtab` file, change the next to last field for both the `ttya` entry and the `ttyb` entry from `y` to `n` (the colon (`:`) acts as the field separator):

- `# vi /etc/saf/zsmon/_pmtab`

5. Reboot the system.

Note – When installing Solaris, set the time zone by selecting a time zone format that conforms to the POSIX.1 format for TZ defined on page 152 and page 153 of the IEEE Std. 1003.1–1990.

Note – The following option must be added to any compiler line:
-D_XOPEN_VERSION=4

Note – The following must be added to any link/load line:
/usr/ccs/lib/values_xpg4.o

TEMPORARY WAIVERS

List below references to any temporary waivers granted by X/Open regarding minor errors in the product referenced above. This should include the X/Open reference and the waiver expiration date. The waivers as granted shall be made available with this document on request.

There are no temporary waivers.

1.1 General Attributes

1.1.1 XPG4 Feature Groups

Question 1: *Which of the following Feature Groups are supported by the implementation?*

XCU4 DEVELOPMENT Option

XCU4 FORTRAN Option

Answer: XCU4 DEVELOPMENT Option

Rationale

Commands and Utilities, Issue 4 states that the system may provide one or more of the Feature Groups listed.

Reference

X/Open CAE Specification, Commands and Utilities, Issue 4, Section 1.2, Conformance and Section 1.3, Status of Interfaces.

1.1.2 XPG4 Development Utilities

Question 2: *Does the implementation support the development utility dis?*

Answer: Yes

Rationale

Commands and Utilities, Issue 4 states that optional utilities listed in Section 1.3.1 on page 2 need not be provided. Section 1.3.1 lists the dis utility as OPTIONAL.

Reference

X/Open CAE Specification, Commands and Utilities, Issue 4, Section 1.2, Conformance and Section 1.3, Status of Interfaces.

1.1.3 XPG4 Unsupportable Utilities

Question 3: *Are the following utilities supported by the implementation?*

Answer:

Utility	Supported
cancel	YES
cu	YES
lpstat	YES
uucp	YES
uulog	YES
uuname	YES
uupick	YES
uuto	YES

Rationale

Commands and Utilities, Issue 4 states that the system need not provide the possibly unsupportable utilities listed.

Reference

X/Open CAE Specification, Commands and Utilities, Issue 4, Section 1.2, Conformance and Section 1.3, Status of Interfaces.

Question 4: *Are the following options to the specified utilities supported by the implementation?*

Answer:

Options	Supported
-*s* option to *ar*	YES
-*E* option to *cc*	YES
-*P* option to *cc*	YES
-*S* option to *cc*	YES

(continued)

Options	Supported
-*m* option to *lp*	YES
-*o* option to *lp*	YES
-*t* option to *lp*	YES
-*w* option to *lp*	YES
-*p* option to *mail*	YES
-*q* option to *mail*	YES
-*r* option to *mail*	YES
-*t* option to *mail*	YES
-*n* option to *pg*	YES
-*z* option to *sort*	NO (See Note)
-*s* option to *ar*	YES
-*r* option to *sum*	YES
+*m[n]* option to *tabs*	YES
-*q* option to *uustat*	YES
-*r* option to *uustat*	YES
-*j* option to *uux*	YES

The support of these options to the specified utilities are an extension to the X/Open CAE Specification standard. These options need not be supported on all XSI-conformant systems.

Rationale
Possibly unsupportable feature.

Reference
X/Open CAE Specification, Commands and Utilities, Issue 4.

Note – *sort* ignores the -*z* option and allocates as much memory as needed to sort the lines. It will not fail if a line longer than `recsz` is encountered.

1.1.4 POSIX.2 Supported Features

Question 5: *Are the following features available on the system?*

Answer:

Macro Name	Meaning	Provided
POSIX2_C_DEV	The implementation supports the C-language development utilities in Annex A of POSIX.2.	Yes
POSIX2_CHAR_TERM	The implementation supports character-based terminals.	Yes
POSIX2_FORT_DEV	The implementation supports the FORTRAN language development utilities in Annex C of POSIX.2.	No
POSIX2_FORT_RUN	The implementation supports the FORTRAN runtime utilities in Annex C of POSIX.2.	No
POSIX2_SW_DEV	The implementation supports the software development utilities in Section 6 of POSIX.2.	Yes
POSIX2_UPE	The implementation supports the User Portability Utilities in Section 5 of POSIX.2.	Yes

Rationale

POSIX.2 states that a POSIX.2 conforming implementation may provide one or more of the following: the User Portability Option, the Software Development Utilities Option, the C-Language Development Utilities Option, the FORTRAN Development Utilities Option, or the FORTRAN Runtime Utilities Option. It also lists POSIX2_CHAR_TERM and POSIX2_LOCALEDEF as the implementation options.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 1.3.

Question 6: *Which terminal types are supported by the implementation?*

Answer: Any terminal for which a user can obtain a TERMINFO description and run through the `tic` utility.

Rationale

The implementation shall document which terminal types it supports and which of the features and utilities are not supported by each terminal.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 2.14.

Question 7: *Does the implementation support moving files across file systems using the mv utility?*

Answer: Yes

Rationale

The mv utility shall perform actions equivalent to the POSIX.1 *rename()* function, called with the following arguments:
(a) The source_file operand is used as the old argument.
(b) The destination path is used as the new argument.

If the links named by new and old are on different file systems, and the implementation does not support links between file systems, the *rename()* function shall return -1 and set errno to [EXDEV].

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 2.14.

POSIX–PART 1: SYSTEM APPLICATION PROGRAM INTERFACE, Section 5.5.3

Question 8: *Does the implementation require write permission for an existing directory in order for mv utility to rename that directory?*

Answer: No

Rationale

The mv utility shall perform actions equivalent to the POSIX.1 *rename()* function, called with the following arguments:

(a) The source_file operand is used as the old argument.

(b) The destination path is used as the new argument.

If write permission is required and is denied for a directory pointed to by the old or new arguments, the *rename()* function shall return -1 and set errno to [EACCES].

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 4.43.

POSIX–PART 1: SYSTEM APPLICATION PROGRAM INTERFACE, Section 5.5.3.

Question 9: *Does the implementation consider it to be an error if a nonprivileged process attempts to rename a file or directory using the mv utility while that file or directory is being used by the system or another process?*

Answer: No

Rationale

The mv utility shall perform actions equivalent to the POSIX.1 *rename()* function, called with the following arguments:

(a) The source_file operand is used as the old argument.

(b) The destination path is used as the new argument.

If the directory named by old or new cannot be renamed because it is being used by the system or another process and the implementation considers this to be an error, the *rename()* function shall return -1 and set errno to [EBUSY].

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 4.43.

POSIX–PART 1: SYSTEM APPLICATION PROGRAM INTERFACE, Section 5.5.3.

Question 10: *Does the implementation support creating hard links across file systems using the ln utility?*

Answer: No

Rationale

The ln utility shall perform actions equivalent to the POSIX.1 *link()* function using source_file as the path1 argument, and the destination path as the path2 argument.

If the link named by new and the file named by existing are on different file systems, and the implementation does not support links between file systems, the *link()* function shall return -1 and set errno to [EXDEV].

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 4.33.

POSIX–PART 1: SYSTEM APPLICATION PROGRAM INTERFACE, Section 5.3.4.

Question 11: *Does the implementation support the floating point conversions in the format string of the printf utility?*

Answer: Yes

Rationale

The e, E, f, g, and G conversion specifications need not be supported in the format string of the format operand of the printf utility.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 4.50.

Question 12: *In the system's handling of Basic Regular Expressions, do the following characters in a subexpression anchor that subexpression?*

Answer:

Feature	Response
The circumflex (^) character	NO
The dollar-sign ($) character	NO

Rationale

The implementation may treat circumflex as an anchor when used as the first character of a subexpression. The implementation may treat a dollar-sign as an anchor when used as the last character of a subexpression.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 2.8.

Question 13: *Does the implementation support the use of numeric signal numbers in the shell's trap command?*

Answer: Yes

Rationale

An implementation may allow numeric signal numbers for the conditions in the trap built-in utility as an extension.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 3.14.13.

1.1.5 POSIX.1 Supported Features

Question 14: *Does the implementation support the POSIX.1 C language interfaces?*

Answer: Yes

Rationale

If the implementation conforms to POSIX.1, all but the shell special built-in utilities can be *exec()*'ed.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 2.3.

1.1.6 Implementation-Dependent Limits Affecting Utilities

Question 15: *What are the minimum values associated with the following symbols?*

Answer:

Macro Name	Meaning	Min. Value
BC_BASE_MAX	Maximum *obase* value allowed by the *bc* utility.	99
BC_DIM_MAX	Maximum number of elements permitted in an array by the *bc* utility.	2048
BC_SCALE_MAX	Maximum scale value allowed by the *bc* utility	99
BC_STRING_MAX	Maximum length of a string constant accepted by the *bc* utility.	1000

(Continued)

Macro Name	Meaning	Min. Value
COLL_WEIGHTS_MAX	Maximum number of weights that can be assigned to an entry of the LC_COLLATE order keyword in the locale definition file.	2
EXPR_NEST_MAX	Maximum number of expressions that can be nested within parentheses by the expr utility.	32
LINE_MAX	Maximum length in bytes including the trailing newline of a utility's input line when the utility is described as processing text files.	2048
RE_DUP_MAX	Maximum number of repeated occurrences of a regular expression permitted when using interval notation.	255

Rationale

The minimum value of these symbols on any XSI conforming system must not be more restrictive than the value of the corresponding POSIX2_ symbols. A specific conforming implementation may provide a higher minimum value than this.

Reference

POSIX–PART 2: SHELL AND UTILITIES, Section 2.13.

Section 1.2: Internationalized System Interfaces

1.2.1 Coded Character Sets

Question 16: *What coded character sets are supported by the implementation?*

Answer: Any character set that can be encoded in EUC: ISO 8859-1, CNS 11643-1992, JIS X0208, KS C 5861-1992, GB23/2-80

Rationale

System Interface Definitions, Issue 4 states that conforming implementations support one or more coded character sets, and that each of these includes the portable character set.

Reference

X/Open CAE Specification, System Interfaces Definitions, Issue 4, Chapter 4, Character Set

Question 17: *What is the implementation's underlying internal codeset?*

Answer: Encoding method uses EUC (Extended UNIX codeset).

Rationale

It is useful to be aware of the underlying codeset of the implementation.

Reference

X/Open CAE Specification, System Interface Definitions, Issue 4, Chapter 4, Character Set.

X/Open Interim UNIX Branding 7

This chapter discusses the compliance of Solaris 2.5 to the questions contained in the X/Open Conformance Statement Questionnaire on Interim UNIX Branding.

X/Open Conformance Statement for SPARC

This section addresses the questionnaire for SPARC based systems.

X/Open Conformance Statement

X/OPEN Conformance Statement Questionnaire

1. Interim UNIX Branding

Product Identification

Product Identification	Solaris for SPARC based systems
Version/Release No.	Solaris 2.4 and on with SPARCompiler C 2.0.1 and on

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance

a. The product is XPG4 branded.

b. The profile registration number is P0034.

c. The product conforms to SVID Edition 3 Base.

d. The product is derived from Novell's operating system technology.

e. The product does qualify as a sublicensed product.

X/Open Conformance Statement for x86

This section addresses the questionnaire for x86 based systems.

X/Open Conformance Statement

X/OPEN Conformance Statement Questionnaire

1. Interim UNIX Branding

Product Identification

Product Identification	Solaris for x86
Version/Release No.	Solaris 2.4 and on with ProCompiler 2.0.1 and on for x86 based systems

If you do not supply this component yourself, please identify below the supplier you reference.

Conformance

a. The product is XPG4 branded.

b. The profile registration number is P0033.

c. The product conforms to SVID Edition 3 Base.

d. The product is derived from Novell’s operating system technology.

e. The product does qualify as a sublicensed product.

POSIX.1 8

Portable Operating System Interface for Computer Environments (POSIX.1)

The IEEE Std 1003.1-1990 Portable Operating System Interface Part 1 (POSIX .1*)* is part of the POSIX series of standards for applications and user interfaces to open systems. POSIX.1 has also been adopted as international standard ISO/IEC 9945-1: 1990 by the International Organization for Standardization/International Electrotechnical Commission. It defines the applications interface to basic system services for input/output, file system access, and process management, using the C programming language, which establishes standard semantics and syntax. Because this interface enables application writers to write portable applications, it has been named POSIX, an acronym for Portable Operating System Interface. POSIX.1 is based on the UNIX operating system and is derived from efforts of the /usr/group Standards Committee. Within this chapter the POSIX.1 standard is referred to in places as “the standard.”

Amending POSIX.1: The IEEE Standards 1003.1b and 1003.1c

IEEE Std 1003.1b–1993 and IEEE Std 1003.1c–1995 are standards that amend POSIX.1 to include extensions in support of real-time and multithreaded applications. The functionality of SunOS 5.5 is intended to provide compliance and support of POSIX.1 as amended by IEEE Stds 1003.1b–1993 and 1003.1c–1995.

8

Scope

To comply with Section 1.3.1.2 (*Documentation*), this chapter describes the behavior of features in the SunOS 5.5 operating system which are described in the standard as *implementation-defined* or for which it is stated that implementations may vary. It does not describe any extensions or enhancements outside the scope of the standard.

Note – Section 2.2.1.2 of the standard defines the term *implementation-defined* as follows: "An indication that the implementation shall define and document the requirements for correct program constructs and correct data of a value or behavior."

The information contained within this chapter does not replace the standard; rather, it serves as an adjunct to the standard for supplying the technical information needed by application developers to write source code within the SunOS 5.5 operating system framework.

C Standard Compliance

The SunOS 5.5 operating system conforms to POSIX.1. The C language compiler and libraries conform to ANSI C.

Audience

This explication is for the experienced C programmer who, when writing an applications program designed to conform to the standard, needs to know the specific behavior of the *implementation-defined* features mentioned in the standard.

Each subsection is prefaced by the appropriate section taken directly from the standard and has the corresponding section number attached to the title.

Note – For maximum portability, applications should not depend upon any particular behavior that is *implementation-defined*.

Notation Used in the Remainder of this Chapter

The following format is used to identify which passage of text is quoted from the standard and which passage of text describes how the SunOS 5.5 operating system implements that area.

- **P.** stands for POSIX.
- **S.** stands for SunOS 5.5.

Section numbers cited in the remainder of this chapter correspond to those of the standard. When creating an application program, this format will help you to quickly locate additional information that you need from the standard.

Implementation-Defined Areas of POSIX.1

POSIX.1 Section 1: General

1.3.1 Implementation Conformance

1.3.1.1 Requirements

P. The conformance document shall define an environment in which an application can be run with the behavior specified by the standard.

S. Solaris 2.5 conforms to IEEE Std 1003.1b-1993 as installed. There are no special procedures required.

1.3.1.2 Documentation

P. The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies.

S. The SunOS 5.5 operating system is a conforming implementation as defined in Section 1.3.1.2 (*Documentation)* of the IEEE Std 1003.1b-1993 *Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface [C Language].*

P. The conformance document may also list international software standards that are available for use by a Conforming POSIX.1 Application.

S. The ANSI X3.159-1989 C Language Standard.

1.3.3.2 C Standard Language Dependent System Support
P. Implementors shall meet the requirements of Section 8 using for reference the C Standard {2}. Implementors shall clearly document the version of the C Standard {2} referenced in fulfilling the requirements of Section 8.

S. The system provides an ANSI C Standard Language Binding as specified by X3.159-1989. This language binding is accessed by specifying either -Xa or -Xc on the cc command line.

POSIX.1 Section 2, Terminology and General Requirements

2.2.2 General Terms

2.2.2.4: appropriate privileges
P. An *implementation-defined* means of associating privileges with a process with regard to the function calls and function call options defined in the standard that need special privileges. There may be zero or more such means.

S. A process whose effective user ID is 0 has all available privileges. Means by which the effective UID may be set to 0 include: logging in as root or any user with a UID equal to zero; issuing the `su command` to change the UID to root or any user with UID equal to zero; successful execution of a file with the S_ISUID bit set and UID equal to zero.

2.2.2.9: character special file
P. One specific type of character special file is a terminal device file, whose access is defined in 7.1. Other character special files have no structure defined by this part of ISO/IEC 9945, and their use is unspecified by this part of ISO/IEC 9945.

S. In addition to terminal device files, the `/dev/ksyms` character special file is available. The `/dev/ksyms` structure is described in the `ksyms`(7) `man` page.

2.2.2.55: parent process ID
P. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an *implementation-defined* system process.

S. If a child process continues to exist after its creator process ceases to exist, the child process is inherited by init. The init process ID is 1.

2.2.2.57: pathname
P. A pathname that begins with two successive slashes may be interpreted in an *implementation-defined* manner.

S. Multiple successive slashes are considered the same as one slash.

2.2.2.68: process lifetime
P. When another process executes a *wait()* or *waitpid()* function for an inactive process, the remaining resources are returned to the system.

S. All resources except the session ID, the process ID and the process group ID are returned to the system.

2.2.2.69: read-only file system
P. A file system that has *implementation-defined* characteristics restricting modifications.

S. A read-only file system does not allow for modification of its files or directories.

2.2.2.83: supplementary group ID
P. Whether a process's effective group ID is included in or omitted from its list of supplementary group IDs is unspecified.

S. A process's effective group ID is included in its list of supplementary group IDs only if the login user ID is a member of the group associated with the effective group ID.

2.3 General Concepts

2.3.1: extended security controls
P. The access control and privilege mechanisms have been defined to allow *implementation-defined* extended security controls.

S. No extended security controls are supported.

2.3.2: file access permissions
P. Implementations may provide additional or alternate file access control mechanisms, or both.

S. There is no additional or optional file access control mechanism.

2.3.5: file times update
P. An implementation may update fields that are marked for update immediately, or may update such fields periodically.

S. The UFS file system updates periodically.

2.4 Error Numbers

P. Implementations may support additional errors not included in this clause, may generate errors included in this clause under circumstances other than those described in this clause, or may contain extensions or limitations that prevent some errors from occurring.

S. In addition to the errors listed in this clause, Solaris supports the following errors:

Error	Error Code	Condition
EBADMSG	77	Message waiting to be read on a data stream is unreadable.
EMULTIHOP	74	Components of path require hopping to multiple remote machines and the file system does not allow it.
ENOLINK	67	Path points to remote machine; link to that machine has been severed or is no longer active.
ENOSR	63	Insufficient streams memory resources available in the system.
EOVERFLOW	79	Value too large to be stored in data type.

[EFAULT]
P. The reliable detection of this error is *implementation-defined*; however, implementations that do detect this condition shall use this value.

S. The functions listed below reliably detect a bad address and return EFAULT when the address is not in a page mapped into the process.

access	chdir	chmod
chown	clock_settime	clock_getres
creat	execl	execle
execlp	execv	execve

execvp	fcntl	fstat
getgroups	link	mkdir
nanosleep	open	read
rmdir	rename	sigpending
sigprocmask	sigsuspend	sigaction
sigwaitinfo	sigtimedwait	sigqueue
stat	tcgetattr	tcsetattr
timer_create	timer_gettime	timer_settime
times	uname	unlink
utime	write	

[EFBIG]
P. The size of a file would exceed an *implementation-defined* maximum file size.

S. The maximum file size is defined by the `setrlimit()` function and can be retrieved by the `getrlimit()` function.

2.5 Primitive System Data Types

`<sys/types.h>`
P. Some data types used by the various system functions are not defined as part of this standard, but are *defined by the implementation*.

S. Additional fundamental data types are:

uchar_t	ushort_t	o_mode_t
uint_t	ulong_t	o_dev_t
caddr_t	daddr_t	o_uid_t
major_t	minor_t	o_gid_t
key_t	o_nlink_t	hostid_t
addr_t	cnt_t	o_ino_t
label_t	paddr_t	use_t
sysid_t	index_t	lock_t
boolean_t	k_sigset_t	k_fltset_t
id_t	o_pid_t	clock_t
wchar_t		

2.6 Environment Description

P. Environment variable *names* used or created by an application should consist solely of characters from the portable filename character set. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names.

S. Any character except NULL and “=” is permitted; however applications should restrict characters to that of the portable filename character set to ensure portability.

2.7 C Language Definitions

2.7.2. POSIX.1 Symbols
P. Implementations, future versions of this part of ISO/IEC 9945, and other standards may define additional feature test macros.

S. Additional defined feature test macros: _XOPEN_SOURCE, _POSIX_C_SOURCE, __REENTRANT and _KERNEL. Use of these macros is described in the *X/Open Portability Guide Issue 3* and IEEE Std 1003.1b–1993.

2.8 Numerical Limits

P. The conformance document shall describe the limit values found in the `<limits.h>` header, stating values, the conditions under which those values may change, and the limits of those variations, if any.

S. `<limits.h>` contains the following magnitude limitations:

Name	Value	Comments
AIO_LISTIO_MAX	Undefined. Value may be configurable in future.	Maximum number of I/O operations in a single list I/O call supported by the implementation.
AIO_MAX	Undefined. Value may be configurable in future.	Maximum number of outstanding asynchronous I/O operations supported by the implementation.
AIO_PRIO_DELTA_MAX	Undefined. Value may be configurable in future.	The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
ARG_MAX	1048320	Maximum length of arguments for

		the exec functions, in bytes, including environment data.
CHILD_MAX	Configurable with minimum value 25.	Maximum number of simultaneous processes per real user ID.
DELAYTIMER_MAX	Undefined. Value may be configurable in future.	Maximum number of timer expiration overruns.
LINK_MAX	32767	Maximum value of a files link count.
LOGIN_NAME_MAX	Undefined. Value may be configurable in future.	Maximum value of a login name.
MAX_CANON	256	Maximum number of bytes in a terminal canonical input line.
MAX_INPUT	512	Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed.
MQ_OPEN_MAX	Undefined. Value may be configurable in future.	The maximum number of open message queue descriptors a process may hold.
MQ_PRIO_MAX	Undefined. Value may be configurable in future.	The maximum number of message priorities supported by the implementation.
NAME_MAX	Undefined. Depends on underlying file system type.	Maximum number of bytes in a file name (not a string length; count excludes a terminating null).
NGROUPS_MAX	16	Maximum number of simultaneous supplementary groups IDs per process.
OPEN_MAX	Default maximum 64 can be raised or lowered by calling setrlimit().	Maximum number of files that one process can have open at one time.
PAGESIZE	Undefined. Depends on system hardware.	Granularity in bytes of memory mapping and process memory locking.
PATH_MAX	1024	Maximum number of bytes in a pathname (not a string length; count excludes a terminating null).
PIPE_BUF	5120	Maximum number of bytes that can be written atomically when writing to a pipe.
PTHREAD_DESTRUCTOR_ITERATIONS	Undefined. Value may be configurable in future.	Maximum number of attempts made to destroy a thread's thread-specific data values on thread exit.
PTHREAD_KEYS_MAX	Undefined. Value may be configurable in future.	Maximum number of data keys that can be created per process
PTHREAD_STACK_MIN	Run the valuated expression	Minimum size in bytes of thread stack storage.

PTHREAD_THREADS_MAX	Undefined. Value may be configurable in future.	Maximum number of threads that can be created per process.
RTSIG_MAX	Undefined. Value may be configurable in future. .	Maximum number of real-time signals reserved for application use in this implementation.
SEM_NSEMS_MAX	Undefined. Value may be configurable in future.	Maximum number of semaphores that a process may have.
SEM_VALUE_MAX	Undefined. Value may be configurable in future.	Maximum value a semaphore may have.
SIGQUEUE_MAX	Undefined. Value may be configurable in future.	Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
STREAM_MAX	Not defined. Depends on OPEN_MAX and number of files open not using fopen().	The number of streams that one process can have open at one time. If defined, it shall have the same value as {FOPEN_MAX} from the C Standard {2}.
SSIZE_MAX	2147483647	The maximum value that can be stored in an object of type ssize_t.
TIMER_MAX	Undefined. Value may be configurable in future.	Maximum number of timers per process supported by the implementation.
TTY_NAME_MAX	Undefined. Value may be configurable in future.	Maximum length of terminal device name.
TZNAME_MAX	Undefined. Time zone name length is limited only by address space.	Maximum number of bytes supported for the name of a time zone. (Not of the TZ variable).

2.9 Symbolic Constants

P. The conformance document shall describe the limit values found in the `<unistd.h>` header, stating values, the conditions under which those values may change, and the limits of those variations, if any.

S. `<unistd.h>` contains the following values:

Name	Value	Comments
_POSIX_CHOWN_RESTRICTED*	Not defined. Depends on underlying file system type.	The use of the chown () function is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its

		supplementary group IDS.
_POSIX_FSYNC	1	fsync() is supported.
_POSIX_JOB_CONTROL	1	Job control is supported.
_POSIX_MAPPED_FILES	1	Mapped files are supported.
_POSIX_MEMLOCK	1	Memory locking is supported.
_POSIX_MEMLOCK_RANGE	1	Memory range locking is supported.
_POSIX_MEMORY_PROTECTION	1	Memory protection is supported.
_POSIX_NO_TRUNC*	Not defined. Depends on underlying file system type.	Pathname components longer than NAME_MAX generate an error.
_POSIX_REALTIME_SIGNALS	1	Real-time signal extension is supported.
_POSIX_SAVED_IDS	1	Saved IDs is supported.
_POSIX_SYNCHRONIZED_IO	1	Synchronized I/O is provided.
_POSIX_TIMERS	1	Clocks and timers are supported.
_POSIX_THREADS	1	Solaris 2.5 supports the Thread option.
_POSIX_THREADS_ATTR_STACKADDR	1	Solaris 2.5 supports the Thread Stack Address Attribute option.
_POSIX_THREADS_ATTR_STACKSIZE	1	Solaris 2.5 supports the Thread Stack Size Attribute option.
_POSIX_THREADS_PRIORITY_SCHEDULING	1	Solaris 2.5 supports the Thread Execution Scheduling option
_POSIX_THREADS_PRIO_INHERIT	Not defined.	
_POSIX_THREADS_PRIO_PROTECT	Not defined.	
_POSIX_THREADS_PROCESS_SHARED	1	Solaris 2.5 supports the Process-Shared Synchronization option.
_POSIX_THREADS_SAFE_FUNCTIONS	1	Solaris 2.5 supports the Thread-Safe Functions option.
_POSIX_VERSION	199506L	Solaris conforms to IEEE Standard 1003.1-1990, as amended by the IEEE Standard for Real-time Extensions approved Sept. 1993 and for Thread Extensions approved June 1995.
_POSIX_VDISABLE*	0	Terminal special characters can be disabled using this character value.

* `_POSIX_CHOWN_RESTRICTED` and `_POSIX_NO_TRUNC` apply to all files on a native SunOS filesystem. `_POSIX_VDISABLE` applies to terminal files.

POSIX.1 Section 3, Process Primitives

3.1.1.2 Process Creation: Description

P. For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the policy and priority settings of the parent process during a *fork()* function. For other scheduling policies, the policy and priority settings on *fork()* are *implementation-defined.*

S. All new child processes inherit the parent's scheduling policy and parameters.

P. The child process has its own copy of the parent's open directory streams. Each open directory stream in the child process may share directory stream positioning with the corresponding directory stream of the parent.

S. Each open directory stream in the child process does not share directory stream positioning with the corresponding directory stream of the parent.

3.1.1.4 Errors

P. For each of the following conditions, if the condition is detected, the *fork()* function shall return -1 and set *errno* to the corresponding value:

[ENOMEM]	The process requires more space than the system is able to supply.

S. The *fork()* function detects the conditions and returns the corresponding *errno* value for [ENOMEM].

3.1.2.2 Execute a File: Description

P. The argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable **PATH** (see 2.6). If this environment variable is not present, the results of the search are *implementation-defined***.**

S. When **PATH** is not set, SunOS 5.5 supplies a default search path:

/usr/sbin:/usr/bin (if the real or effective UID is root.)

/usr/bin: (if neither the real nor the effective UID is that of root.)

P. For the SCHED_FIFO and SCHED_RR scheduling policies, the policy and priority settings shall not be changed by a call to an *exec* function. For other scheduling policies, the policy and priority settings on exec are *implementation-defined***.**

S. For all scheduling policies, the policy and scheduling parameters are not changed by a call to an *exec* function.

P. Any outstanding asynchronous I/O operations may be canceled.Those asynchronous I/O operations which are not canceled shall complete as if the *exec* function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the *exec* function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the *exec* function be affected by the presence of outstanding asynchronous I/O operations at the time the exec function is called. Whether any I/O is canceled, and which I/O may be canceled upon exec is *implementation-defined.*

S. Any cancelable asynchronous I/O operations are canceled.

3.1.2.4 Execute a File: Errors

P. If any of the following conditions occur, the *exec* functions shall return -1 and set *errno* to the corresponding value:

[EACCES]	Search permission is denied for a directory listed in the path prefix of the new process image file, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.

S. Only regular files are supported by the *exec* function.

P. For each of the following conditions, if the condition is detected, the *exec* functions shall return -1 and return the corresponding value to *errno*:

[ENOMEM]	The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

S. The *exec* functions detect the conditions and return the corresponding *errno* value for [ENOMEM].

3.2.1.2 Wait for Process Termination: Description

wait()
P. An *implementation may define* additional circumstances under which *wait()* or *waitpid()* reports status. In these cases the interpretation of the reported status is *implementation-defined.*

S. A child that is being traced stops because it has reached a break point. WIFSTOPPED (status) will be true and WSTOPSIG (status) will yield the signal that caused the process to stop.

If a child that was formerly stopped by Job Control was continued, WIFCONTINUED (status) will be true.

The above will not be true unless the process is tracing a child. See the `proc(4) man` page for more information.

3.2.2.2 Terminate a Process: Description

exit()
P. Children of a terminated process shall be assigned a new parent process ID, corresponding to an *implementation-defined* system process.

S. The child’s parent process ID becomes 1 which is the process ID of the init process.

P. Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations which are not canceled shall complete as if the _exit() operation had not yet occurred, but any associated signal notifications shall be suppressed. The _exit() operation itself may or may not block awaiting such I/O completion. Whether any I/O is canceled, and which I/O may be canceled upon _exit(), is *implementation-defined.*

S. Any cancelable asynchronous I/O operations are canceled.

3.3.1.1 Signal Names

`<signal.h>`
P. An *implementation may define* additional signals that may occur in the system.

S. The additional signals generated are:

SIGILL	SIGTRAP
SIGEMT	SIGBUS
SIGSYS	SIGPWR
SIGWINCH	SIGURG
SIGPOLL	SIGVTALRM
SIGPROF	SIGXCPU
SIGXFSZ	SIGIOT
SIGLOST	SIGIO
SIGFREEZE	SIGTHAW

P. It is *implementation-defined* whether the real-time signal behavior specified in this section—specifically, the queueing of signals and the passing of application defined values—is supported for the signals defined in Table 3-1, Table 3-2 or Table 3-3 [of the standard].

S. The passing of application-defined values is supported for all signals. The queueing of signals is supported for all signals generated via `sigqueue()` or requested via a `sigevent.sigev_notify` value of SIGEV_SIGNAL.

3.3.1.2 Signal Generation and Delivery

Signals
P. If a subsequent occurrence of a pending signal is generated, it is *implementation-defined* as to whether the signal is delivered more than once.

S. Subsequent occurrences of signals are delivered more than once if the subsequent signal was generated via *sigqueue()* or requested via a sigevent.sigev_notify value of SIGEV_SIGNAL.

P. An *implementation shall document* any conditions not specified by this standard under which the implementation generates signals.

S. Conditions under which these additional signals are generated are:

Signal	Condition
SIGTRAP	Trace/breakpoint Trap
SIGWINCH	Window size change
SIGEMT	Emulation trap
SIGURG	Urgent socket condition
SIGPOLL	Pollable event
SIGBUS	Bus error
SIGVTALRM	Virtual timer expired
SIGILL	Illegal instruction
SIGSYS	Bad system call
SIGPROF	Profiling timer expired
SIGXCPU	CPU time limit exceeded
SIGPWR	Power fail/restart
SIGXFSZ	File size limit exceeded
SIGIO	On asynchronous I/O
SIGLOST	When a lock is broken. (See lockd(8))
SIGFREEZE	Checkpoint freeze
SIGTHAW	Checkpoint thaw

P. If *sigev_notify_attributes* is **NULL**, the behavior shall be as if the thread were created with the `detachstate` attribute set to PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a `detachstate` attribute of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this thread is *implementation-defined.*

S. The signal mask is indeterminate.

P. Either the implementation shall support the behavior specified above or the implementation shall treat the SIGEV_THREAD value as an error.

S. *timer_create()* does not support this value of `sigev_notify_attributes`.

3.3.1.3 Signal Actions

P. The following values are defined for si_code:

SI_USER	The signal was sent by the *kill()* function. The implementation may set si_code to SI_USER if the signal was sent by the *raise()* or *abort()* functions defined in the C Standard {2} or any

	similar functions provided as implementation extensions.
SI_QUEUE	The signal was sent by the *sigqueue()* function.
SI_TIMER	The signal was generated by the expiration of a timer set by *timer_settime()*.
SI_ASYNCIO	The signal was generated by the completion of an asynchronous I/O request.
SI_MESGQ	The signal was generated by the arrival of a message on an empty message queue.

If the signal was not generated by one of the functions or events listed above, the si_code shall be set to an *implementation-defined* value that is not equal to any of the values defined above.

S. SunOS defines other values of si_code for particular signals. Symbols for these values are described in the `siginfo (5)` manual pages. Due to compilation namespace requirements, these symbols are not defined in the POSIX compilation environment, and hence are not available to Strictly Conforming Applications. Aliases for the values of si_code that begin with “SI” are not currently available.

3.3.2.2 Send a Signal to a Process: Description

kill()
P. An implementation that provides extended security controls may impose further *implementation-defined* restrictions on the sending of signals, including the null signal.

S. Extended security controls that impose further restrictions on the sending of signals are not provided.

3.3.3.4 Manipulate Signal Sets: Errors

P. For each of the following conditions, if the condition is detected, the *sigaddset()*, *sigdelset()*, and *sigismember()* functions shall return -1 and set *errno* to the corresponding value:

[EINVAL]	The value of the *signo* argument is an invalid or unsupported signal number.

S. The *sigaddset()*, *sigdelset()*, and *sigismember()* functions all detect [EINVAL].

3.3.4.2 Examine and Change Signal Action: Description

P. If SA_SIGINFO is not set in sa_flags, then the disposition of subsequent occurrences of sig when it is already pending is *implementation-defined*; and the signal-catching function shall be invoked with a single argument.

S. If SA_SIGINFO is not set in sa_flags, subsequent occurrences of sig, when it is already pending and queued, are quietly discarded.

3.3.6.4 Examine Pending Signals: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *sigpending()* function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The *sigpending()* function detects [EFAULT].

3.3.8.2 Synchronously Accept a Signal: Description

P. If prior to the call to *sigwait()* there are multiple pending instances of a single signal number, it is *implementation-defined* whether upon successful return there are any remaining pending signals for that signal number.

S. Clears only the first instance of the signal. Other instances are left pending.

POSIX.1 Section 4, Process Environment

4.2.4.4 Get User Name: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *getlogin()* function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The system detects no errors for *getlogin()*.

4.4.1.2 Get System Name: Description

`<sys/utsname.h>`
P. The structure *utsname* is defined in the header `<sys/utsname.h>`, and contains at least the members shown in Table 4-1 of the standard. (Refer to the standard for the table members.)

Each of these data items is a null-terminated character array. The format of each member is *implementation-defined.*

S. The format of the members found in `<sys/utsname.h>` for *utsname* is type *char* [257].

4.5.1.4 Get System Time: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *time()* function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No additional errors are detected for the *time()* function.

4.6.1.4 Environment Variables: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *getenv()* function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the getenv() function.

4.7.1.4 Generate Terminal Pathname: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *ctermid()* function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the *ctermid()* function.

4.7.2.4 Determine Terminal Device Name: Errors

P. This part of ISO / IEC 9945 does not specify any error conditions that are required to be detected for the t*tyname()* or *isatty()* functions. Some errors may be detected under conditions that are unspecified by this part of ISO / IEC 9945.

S. No error conditions are detected for the *ttyname()* or *isatty()* functions.

POSIX.1 Section 5, Files and Directories

5.1.1 Format of Directory Entries

P. The internal format of directories is unspecified.

S. For each directory a link count is maintained. This is the total number of directories that are listed in the directory, including "." and "..".

5.1.2.4 Directory Operations: Errors

P. For each of the following conditions, when the condition is detected, the *opendir()* function shall return a value of **NULL** and set *errno* to the corresponding value:

[EMFILE]	Too many file descriptors are currently open for the process.
[ENFILE]	Too many file descriptors are currently open in the system.

S. The *opendir()* function detects the conditions and returns the corresponding *errno* values for both [EMFILE] and [ENFILE].

P. For each of the following conditions, when the condition is detected, the *readdir()* function shall return a value of **NULL** and set *errno* to the corresponding value:

[EBADF]	The dirp argument does not refer to an open directory system.

S. If the dirp argument passed to *readdir()* does not point to an open directory stream, Solaris returns a **NULL** pointer and set *errno* to [EBADF].

P. For each of the following conditions, when the condition is detected, the *closedir()* function shall return a value of -1 and set *errno* to the corresponding value:

[EBADF]	The dirp argument does not refer to an open directory stream.

S. For the *closedir()* function, Solaris may detect the condition and set *errno* value to [EBADF].

5.2.2.4 Get Working Directory Pathname: Errors

P. For each of the following conditions, if the condition is detected, the *getcwd()* function shall return a value of NULL and set *errno* to the corresponding value:

[EACCES]	Read or search permission was denied for a component of the pathname.

S. For the *getcwd()* function, Solaris detects the conditions and returns the corresponding *errno* value for [EACCES].

5.3.1.2 Open a File: Description

O_CREAT
P. The file's group ID shall be set to the group ID of the directory in which the file is being created or to the effective group ID of the process.

S. When O_CREAT is set in oflag and bits in mode other than the file permission bits are set, the files group ID is set to the group ID of the parent directory if the S_ISGID bit is set in the directory in which the file is being created. If the S_ISGID bit is set in the parent directory, the group ID of the file is inherited from the parent directory; otherwise it is set to the group ID of the calling process.

O_TRUNC
P. If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, it shall be truncated to zero length and the mode and owner shall be unchanged by this function call. O_TRUNC shall have no effect on FIFO special files or directories. Its effect on other file types is *implementation-defined*. The result of using O_TRUNC with O_RDONLY is undefined.

S. O_TRUNC has no effect on other file types.

5.3.3.2 Set File Creation Mask: Description

umask()
P. The *umask()* routine sets the process's file mode creation mask to *cmask* and returns the previous value of the mask. Only the file permission bits of *cmask* are used; the meaning of the other bits is *implementation-defined*.

S. The implementation ignores all but the file permission bits.

5.3.4.2 Link to a File: Description

P. The *existing* argument shall not name a directory unless the user has appropriate privileges and the implementation supports using *link()* on directories.

S. Linking of directories is supported if the user has appropriate privileges.

P. The implementation may require that the calling process has permission to access the existing file.

S. Solaris does not require that the calling process have permission to access the existing file when linking files.

5.4.1.2 Make a Directory: Description

mkdir()
P. When bits in `mode` other than the file permission bits are set, the meaning of these additional bits is *implementation-defined*.

S. The implementation ignores all but the file permission bits.

P. The directory's group ID shall be set to the group ID of the directory in which the directory is being created or to the effective group ID of the process.

S. A new directory's group ID is set to the group ID of the parent directory when the S_ISGID bit is set in the parent directory; otherwise it is set to the group ID of the calling process. The newly created directory inherits the S_ISGID bit.

5.4.2.2 Make a FIFO Special File: Description

mkfifo()
P. When bits in `mode` other than the file permission bits are set, the meaning of these additional bits is *implementation-defined.*

S. The implementation ignores all but the file permission bits.

P. The group ID of the FIFO shall be set to the group ID of the directory in which the FIFO is being created or to the effective group ID of the process**.**

S. If the S_ISGID bit is set in the parent directory, the group ID of the FIFO is inherited from the parent directory; otherwise it is set to the group ID of the calling process.

5.5.1.2 Remove Directory Entries: Description

P. The *path* argument shall not name a directory unless the process has appropriate privileges and the implementation supports using *unlink()* on directories.

S. *unlink()* is supported if the user has the appropriate privileges.

5.5.1.4 Remove Directory Entries: Errors

P. If any of the following conditions occur, the *unlink()* function shall return -1 and set *errno* to the corresponding value:

[EBUSY]	The directory named by the *path* argument cannot be unlinked because it is being used by the system or another process and the implementation considers this to be an error.

S. *unlink()* supports detection of [EBUSY].

5.5.2.2 Remove a Directory: Description

rmdir()
P. If the named directory is the root directory or the current working directory of any process, it is unspecified whether the function succeeds or whether it fails and sets *errno* to [EBUSY].

S. If an attempt is made to *rmdir()* the current working directory of the process, the system returns [EBUSY].

5.5.2.4 Remove a Directory: Errors

P. If any of the following conditions occur, the *rmdir*() function shall return -1 and set *errno* to the corresponding value:

[EBUSY]	The directory named by the path argument cannot be removed because it is being used by another process and the implementation considers this to be an error.

S. If the directory indicated in the call to *rmdir*() is a mount point for a mounted file system, *rmdir*() sets *errno* to [EBUSY] and returns -1.

5.5.3.2 Rename a File: Description

P. Write access permission is required for the directory containing *old* and the directory containing *new*. If the *old* argument points to the pathname of a directory, write access permission may be required for the directory named by *old*, and, if it exists, the directory named by *new*.

S. In a call to *rename()*, if the *old* argument points to the pathname of a directory, write access permission is not required for the directory named by *old* and if it exists, for the directory named by *new*.

5.5.3.4 Rename a File: Errors

P. If any of the following conditions occur, the *rename()* function shall return -1 and set *errno* to the corresponding value:

[EBUSY]	The directory named by *old* or *new* cannot be renamed because it is being used by the system or another process and the implementation considers this to be an error.

S. [EBUSY] is returned only if the *new* directory is a mount point for a mounted file system.

5.6.1.2 File Characteristics: File Modes

`<sys/stat.h>`
P. Implementations may `OR` other *implementation-defined* bits into S_IRWXU, S_IRWXG, and S_IRWXO, but they shall not overlap any of the other bits defined in this standard.

S. The implementation also provides a bit identified by S_ISVTX. For a directory, this bit determines whether or not an unprivileged user may delete or rename another user's files from that directory (refer to `chmod(2)` for other files types).

5.6.2.2 Get File Status: Description

stat(), fstat()
P. An implementation that provides additional or alternate file access control mechanisms may, under *implementation-defined* conditions, cause the *stat()* and *fstat()* functions to fail.

S. No other conditions cause these functions to fail.

5.6.3.4 Check File Accessibility: Errors

P. For each of the following conditions, if the condition is detected, the *access()* function shall return -1 and set *errno* to the corresponding value:

[EINVAL]	An invalid value was specified for `amode`.

S. For the *access()* function, Solaris detects the condition and returns the corresponding *errno* value for [EINVAL].

5.6.4.2 Change File Modes: Description

P. Additional *implementation-defined* restrictions may cause the S_ISUID and S_ISGID bits in `mode` to be ignored.

S. If the process has access permissions, there are no implementation-defined conditions under which this would be denied.

P. The effect on file descriptors for files open at the time of the *chmod()* or *fchmod()* function is *implementation-defined.*

S. Access permissions for open file descriptors that refer to files on local (UFS) mounted file systems are not affected by *chmod()* or *fchmod()*. Access permissions for descriptors referring to files on other file systems may change as a result of a successful *chmod()* or *fchmod()* call.

5.6.5.2 Change Owner and Group of a File: Description

chown()
P. If the *path* argument refers to a regular file, the set-user-ID (S_ISUID) and set-group-ID (S_ISGID) bits of the file mode shall be cleared upon successful return from *chown()*, unless the call is made by a process with appropriate privileges, in which case it is *implementation-defined* whether those bits are altered.

S. The S_ISUID and S_ISGID bits of the file mode remain unaltered when a call is made by a process with the appropriate privilege.

5.6.5.4 Change Owner and Group of a File: Errors

P. For each of the following conditions, if the condition is detected, the *chown()* function shall return -1 and set *errno* to the corresponding value:

[EINVAL]	The owner of group ID supplied is invalid and not supported by the implementation.

S. The *chown()* function does not detect [EINVAL].

5.7.1.4 Get Configurable Pathname Variables: Errors

P. If any of the following conditions occur, the *pathconf()* function shall return -1 and set *errno* to the corresponding value:

[EINVAL]	The value of `name` is invalid.

S. For the *pathconf()* function, Solaris does detect the condition and returns the corresponding *errno* value for [EINVAL].

P. For each of the following conditions, if the condition is detected, the *pathconf()* function shall return -1 and set *errno* to the corresponding value:

[EACCES]	Search permission is denied for a component of the path prefix.
[ENAMETOOLONG]	The length of the `path` argument exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.
[ENOENT]	The named file does not exist, or the `path` argument points to an empty string.
[ENOTDIR]	A component of the path prefix is not a directory.

S. For the *pathconf()* function, Solaris detects the conditions and returns the corresponding *errno* value for [EACCES], [ENAMETOOLONG], [ENOENT] and [ENOTDIR].

P. For each of the following conditions, if the condition is detected, the *fpathconf()* function shall return -1 and set *errno* to the corresponding value:

[EBADF]	The fildes argument is not a valid file descriptor.
[EINVAL]	The implementation does not support an association of the variable name with the specified file.

S. For the *fpathconf()* function, Solaris detects the conditions and returns the corresponding *errno* value for [EBADF] and [EINVAL].

POSIX.1 Section 6, Input and Output Primitives

6.3.1.2 Close a File: Description

P. When there is an outstanding cancelable asynchronous I/O operation against fildes when *close()* is called, that I/O operation may be canceled. An I/O operation which is not canceled completes as if the *close()* operation had not yet occurred. All operations which are not canceled shall complete as if the *close()* blocked until the operations completed. The *close()* operation itself need not block awaiting such I/O completion. Whether any I/O operation is canceled, and which I/O operation may be canceled upon *close()*, is *implementation-defined.*

S.The Asynchronous Input and Output option is not supported in Solaris 2.5; hence, there is no implementation-specific behavior.

6.4.1.2 Read from a File: Description

read()
P. If a *read()* is interrupted by a signal after it has successfully read some data, either it shall return -1 with *errno* set to [EINTR], or it shall return the number of bytes read.

S. If a *read()* is interrupted by a signal after is has successfully read some data, it returns the number of bytes read.

P. If the file refers to a device special file, the result of subsequent *read()* requests after a *read()* has returned an EOF indication, is *implementation-defined*.

S. The result of this request is device dependent for standard tty devices. See Section 4 of the *SunOS 5.5 Reference Manual* for more information.

P. If the value of *nbyte* is greater than {SSIZE_MAX}, the result is *implementation-defined*.

S. Given a valid buffer, *nbyte* bytes will be transferred.

6.4.2.2 Write to a File: Description

write()
P. If *write()* is interrupted by a signal after it successfully writes some data, either it shall return -1 with *errno* set to [EINTR], or it shall return the number of bytes written.

S. If *write()* is interrupted by a signal after it successfully writes some data, it returns the number of bytes read.

P. If the value of *nbyte* is greater than {SSIZE_MAX}, the result is *implementation-defined*.

S. The write will not succeed; it returns –1 and sets *errno* to [EINVAL].

6.5.2.2 File Control: Description

P. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, the *fcntl()* function shall fail with an [EDEADLK] error.

S. The *fcntl()* function detects [EDEADLK].

6.5.3.2 Reposition Read/Write File Offset: Description

lseek()
P. Some devices are incapable of seeking. The behavior of the *lseek()* function on such devices is *implementation-defined*.

S. On such devices, *lseek()* returns -1 with *errno* set to [EINVAL].

6.6 File Synchronization

P. The hardware characteristics upon which the implementation relies to assure that data successfully transferred for synchronized I/O operations are *implementation-defined*.

S. The data is considered to be successfully transferred when the device driver operation completes successfully.

6.6.1.2 Synchronize a File's State: Description

P. The *fsync()* function can be used by the application to indicate that all data for the open file description named by *fildes* is to be transferred to the storage device associated with the file described by *fildes*, in an *implementation-defined* manner.

The *conformance document shall include* sufficient information for the user to determine whether it is possible to configure an application and installation to ensure that the data is stored with the degree of required stability for the intended use.

S. For files in a ufs filesystem, the physical transfer to the underlying device must successfully complete.

6.7.1.1 Data Definitions for Asynchronous Input and Output: Asynchronous I/O Control Block

P. Under *implementation-defined* circumstances, such as operation on a multiprocessor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed; *the implementation shall document* under what circumstances the ordering restriction may be relaxed.

S. The Asynchronous Input and Output option is not supplied in Solaris; hence, there is no implementation-specific behavior.

P. The relative priority of asynchronous I/O and synchronous I/O is *implementation-defined*. If POSIX_PRIORITIZED is defined, *the implementation shall define* for which files I/O prioritization is supported.

S. The Asynchronous Input and Output option is not supplied in Solaris; hence, there is no implementation-specific behavior.

6.7.7.2 Cancel Asynchronous I/O Request: Description

aio_cancel()
P. It is *implementation-defined* which operations are cancelable.

S. The Asynchronous Input and Output option is not supplied in Solaris; hence, there is no implementation-specific behavior.

POSIX.1 Section 7, Device- and Class-Specific Functions

7.1 General Terminal Interface

terminal interface
P. It is *implementation-defined* whether this interface supports network connections or synchronous ports or both. The conformance document shall describe which device types are supported by these interfaces.

S. SunSoft supports these interfaces for terminal devices, terminal multiplexers, and terminal pseudo-devices.

7.1.1.3 The Controlling Terminal

controlling terminal
P. The controlling terminal for a session is allocated by the session leader in an *implementation-defined* manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the O_NOCTTY option, it is i*mplementation-defined* whether the terminal becomes the controlling terminal of the session leader.

S. If a session leader has no controlling terminal and opens a terminal device file that is not already associated with a session without using the O_NOCTTY option, the terminal then becomes the controlling terminal of the session leader.

7.1.1.5 Input Processing and Reading Data

input queue
P. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded is *implementation-defined*.

S. If the data in the driver's input queue exceeds {MAX_INPUT}, all the characters saved in the stream up to that point are discarded without notice. However, if IMAXBEL is set and the data in the driver input queue exceeds {MAX_INPUT}, the ASCII BEL character is echoed. Further input will not be stored, and any input already present in the input stream is not disturbed.

7.1.1.6 Canonical Mode Input Processing

{MAX_CANON}
P. If {MAX_CANON} is defined for this terminal device, it is a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is *implementation-defined*.

S. If the data in the line discipline buffer exceeds {MAX_CANON} in the canonical mode and IMAXBEL is not set, all the characters saved in the buffer up to that point are discarded without any notice. However, if IMAXBEL is set and the data in the line discipline buffer exceeds {MAX_CANON}, the ASCII BEL character is echoed. Further input will not be stored, and any input already present in the input stream is not disturbed.

7.1.1.7 Noncanonical Mode Input Processing

MIN
P. If MIN is greater than {MAX_INPUT}, the response to the request is *implementation-defined*.

S. The maximum value that can be stored for MIN in c_cc [VMIN] is 255, which is less than {MAX_INPUT} (512). The MIN value can never exceed {MAX_INPUT}.

7.1.1.8 Writing Data and Output Processing

P. The *implementation may provide* a buffering mechanism; as such, when a call to write() completes, all of the bytes written have been scheduled for transmission to the device, but the transmission will not necessarily have completed.

S. Solaris provides a buffering mechanism for a *write()* to a terminal device. The *write()* system call may complete and return a value to the user program, but the data sent downstream may flow control on one or more streams modules.

7.1.1.9 Special Characters

START, STOP
P. It is *implementation-defined* whether the START and STOP characters can be changed.

S. The START and STOP characters can be changed.

IEXTEN
P. A special character is recognized not only by its value, but also by its context; for example, an implementation may define multi-byte sequences that have a meaning different from the meaning of bytes when considered individually. Implementations may also define additional single-byte functions. These *implementation-defined* multibyte or single byte functions are recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in the subclass (7.1.1.9).

S. SunOS 5.5 does not recognize any multibyte input control sequences. The following single-byte special characters are recognized when IEXTEN is set:

WERASE	Erase last word typed
REPRINT	Reprint the current input
DISCARD	Discard output
LNEXT	Ignore any special meaning of the next character typed

7.1.2.2 Input Modes

c_iflag
P. In contexts other than asynchronous serial data transmission the definition of a break condition is *implementation-defined*.

S. The break condition is not defined for contexts other than asynchronous serial data.

P. The precise conditions under which STOP and START characters are transmitted are *implementation-defined*.

S. A STOP character is transmitted when the input data exceeds the high water mark of the queue. A START character is transmitted when the input data falls below the low water mark of the queue. If it is not a STREAMS device, the results are device-dependent.

P. The initial input control value after *open()* is *implementation-defined*.

S. The initial setting of the input mode flag is configurable. For more information, see the `termio(7) man` page.

7.1.2.3 Output Modes

P. If OPOST is set, output data is processed in an *implementation-defined* fashion so that lines of text are modified to appear appropriately on the terminal device, otherwise characters are transmitted without change.

S. The SunOS 5.5 operating system supports the following output control mode masks which are enabled by OPOST:

OLCUC	Map lower case to upper case on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select new line delays:
NL0	No new line delay.
NL1	
CRDLY	Select carriage-return delays:
CR0	No carriage return delay.

CR1	
CR2	
CR3	
TABDLY	Select horizontal-tab delays or expansion:
TAB0	No horizontal tab delay.
TAB1	
TAB2	
TAB3	
XTABS	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	No backspace delay.
BS1	
VTDLY	Select vertical-tab delays:
VT0	No vertical tab delay.
VT1	
FFDLY	Select form-feed delays:
FF0	No form feed delay.
FF1	

open()
P. The initial output control value after *open()* is *implementation-defined*.

S. The initial setting for the output control flag oflag is configurable. For more information, see the termio(7) man page.

7.1.2.4 Control Modes

open()
P. The initial hardware control value after *open()* is *implementation-defined*.

S. The initial hardware control value after *open()* is configurable. For more information, see the `termio(7) man` page.

7.1.2.5 Local Modes

IEXTEN
P. If IEXTEN is set, *implementation-defined* functions shall be recognized from the input data.

S. If IEXTEN and ICANON are set, the WERASE, REPRINT, DISCARD, and LNEXT functions are recognized from the input data.

P. It is *implementation-defined* how IEXTEN being set interacts with ICANON, ISIG,IXON, or IXOFF. If IEXTEN is not set, then implementation-defined functions shall not be recognized, and the corresponding input characters shall be processed as described for ICANON, ISIG, IXON, and IXOFF.

S. IXON, ISIG, and IXOFF flags are processed as they are defined in the standard, when IEXTEN is on or off.

In addition to the local mode masks listed in the standard, the SunOS 5.5 operating system supports the following functions:

XCASE	Canonical upper/lower presentation
ECHOCTL	Echo control characters as '^C', delete character as '^?'.
ECHOPRT	Echo erase character as character erased.
ECHOKE	BSSP_BS erase entire line on line kill.
FLUSHO	Output is being flushed.
PENDIN	Retype pending input at next read or input character.

P. The initial control value after open() is *implementation-defined.*

S. The initial setting for the local mode flag *lflag* is configurable. For more information, see the `termio(7) man` page.

7.1.2.6 Special Control Characters

P. The initial values of all control characters are *implementation-defined.*

S. The initial values of control characters are configurable and are set when the system boots. See the `termio(7) man` page for more information.

7.1.3.4 Baud Rate Functions: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *cfgetispeed(), cfgetospeed(), cfsetispeed()* or *cfsetospeed()* functions. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The *cfgetispeed(), cfgetospeed() cfsetispeed()* and *cfsetospeed()* functions do not return any additional errors.

7.2.1.2 Get and Set State: Description

P. If the input and output baud rates differ and are a combination that is not supported, neither baud rate is changed.

S. Differing input and output baud rates are not supported.

7.2.2.2 Line Control Functions: Description

tcsendbreak(), tcdrain(), tcflush(), tcflow()
P. If the terminal is not using asynchronous serial data transmission, it is *implementation-defined* whether the *tcsendbreak()* function sends data to generate a break condition (as *defined by the implementation*) or returns without taking any action.

S. On non-asynchronous transmissions, *tcsendbreak()* does not send a break; it simply returns.

tcsendbreak()
P. If `duration` is not zero, it shall send zero-valued bits for an *implementation-defined* period of time.

S. For a delay of $n \neq 0$, *tcsendbreak()*is equivalent to *tcdrain()*.

POSIX.1 Section 8, Language-Specific Services for the C Programming Language

8.1.1 Referenced C Language Routines, Extensions to Time Functions

TZ
P. If **TZ** is of the first format (i.e., if the first character is a colon), the characters following the colon are handled in an *implementation-defined* manner.

S. The string following the colon refers to the file `/usr/share/lib/zoneinfo/`<*string*>. This file contains a timezone specification.

8.1.2.2 Extensions to setlocale(): Description

setlocale()
P. In addition to the value for “category” specified in the standard, the *implementation may define* additional categories.

S. In addition to the categories (environment variables) described in the standard, the SunOS 5.5 operating system supports the following:

LC_MESSAGES	Allows for display of alternate message texts

P. If no nonnull environment variable ($LC_ALL, $LANG, or the environment variable corresponding to the category being set) is present to supply a value for “locale” it is implementation-defined whether *setlocale()* sets the specified locale category to a systemwide default value or to “C or to POSIX”.

S. The default locale is “C”.

P. The possible actual values of the environment variables are *implementation-defined* and should appear in the system documentation.

S. The supported locales are “C”,“POSIX”, “de”,“fr”,“it”, “ja”, “japanese” and “sv”. The locale name “iso_8859_1” contains only the LC_CTYPE category, and is not an appropriate value for LC_ALL or the LANG environment variable. Other locale names may be available due to the addition of further Sun or third-party packages.

8.2.2.4 Open a Stream on a File Descriptor: Errors

P. This part of ISO/IEC does not specify any error conditions that are required to be detected for the *fdopen()* function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The *fdopen()* function detects no errors.

8.2.3 Interactions of Other File-Type C Functions

P. (5) Implementations shall assure that an application, even one consisting of several processes, shall yield correct results (no data is lost or duplicated when writing, all data is written in order, except as requested by seeks) when the rules above are followed, regardless of the sequence of handles used. When these rules are followed, it is *implementation-defined* whether, and under what conditions, all input is seen exactly once.

S. When applications follow the rules specified, all input is seen exactly once.

8.2.7.4 Stdio with Explicit Client Locking: Errors

P. Some errors may be detected under *implementation-defined* conditions, or as defined by C Standard {2}.

S. The following four functions do not set any error conditions:

getc_unlocked()
getchar_unlocked()
putc_unlocked()
putchar_unlocked()

8.3.2.2 Set Time Zones: Description

tzset()
P. If **TZ** is absent from the environment, *implementation-defined* default time zone information shall be used.

S. If **TZ** is absent from the environment, then time zone information behaves as though **TZ** were set to localtime.

8.3.4.4 Find String Token: Errors

P. Some errors may be detected under *implementation-defined* conditions.

S. Solaris 2.5 does not set any error conditions.

8.3.5.4 ASCII Time Representation: Errors

P. Some errors may be detected under *implementation-defined* conditions, or as defined by C Standard {2}.

S. Solaris 2.5 does not set any error conditions.

8.3.6.4 Current Time Representation: Errors

P. Some errors may be detected under *implementation-defined* conditions, or as defined by C Standard {2}.

S. Solaris 2.5 does not set any error conditions.

8.3.7.4 Coordinated Universal Time: Errors

P. Some errors may be detected under *implementation-defined* conditions.

S. Solaris 2.5 does not set any error conditions.

8.3.8.4 Local Time: Errors

P. Some errors may be detected under *implementation-defined* conditions.

S. Solaris 2.5 does not set any error conditions.

POSIX.1 Section 9, System Databases

9.1 System Databases

`/etc/passwd`
P. If the initial user program field is null, the system default is used.

S. If the user program field is null, the default program is `/usr/bin/sh`.

P. If the initial working directory field is null, the interpretation of that field is *implementation-defined*.

S. If the field is empty, the login fails.

9.2.1.4 Group Database Access: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the *getgrgid()* or *getgrnam()* functions. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the *getgrgid()* or *getgrnam()* functions.

9.2.2.4 User Database Access: Errors

P. This part of ISO / IEC 9945 does not specify any error conditions that are required to be detected for the *getpwuid()* or *getpwnam()* functions. Some errors may be detected under conditions that are unspecified by this part of ISO / IEC 9945.

S. No errors conditions are detected for the *getpwuid()* or *getpwnam ( )* functions.

POSIX.1 Section 10, Data Interchange Format

10.1 Archive/Interchange File Format

P. The *format-creating utility* is used to translate from the file system to the formats defined in this clause. The format-reading utility is used to translate from the formats defined in this clause to a file system. The interface to these utilities, including their name or names, is *implementation-defined.*

S. The `cpio` utility, when used with certain options, can be used to create and read these formats. For more information, see the `cpio` `man` page.

10.1.1 Extended `tar` Format

P. If an implementation supports the use of characters outside the portable filename character set in names for files, users, and groups, one or more *implementation-defined* encodings of these characters shall be provided for interchange purposes.

S. The use of all 8-bit characters is supported (except **NULL** and '`/`') in names for files, and all 8-bit characters (except **NULL** and colon) in names for users and / or groups. Characters are used in filenames exactly as they are read from the archive.

P. If a file name is found on the medium that would create an invalid file name, the implementation shall define if the data from the file is stored on the file hierarchy and under what name it is stored.

S. Any name that can be stored in a `tar` archive is interpreted as a valid file name.

10.1.2.1 Header

P. *c_rdev* shall contain *implementation-defined* information for character or block special files.

S. The *c_rdev* field contains devmajor / devminor device numbers. It is a 6- digit octal number and calculated as the major device number left-shifted by 8 ORed with the minor device number.

10.1.2.2 File Name

P. If a file name is found on the medium that would create an invalid pathname, the *implementation shall define* if the data from the file is stored on the file hierarchy and under what name it is stored.

S. All names will be legal in the file hierarchy.

P. If an implementation supports the use of characters outside the portable filename character set in names for files, users, and groups, one or more *implementation-defined* encodings of these characters shall be provided for interchange purposes.

S. The use of all 8-bit characters is supported (except **NULL** and '`/`') in names for files, and all 8-bit characters (except **NULL** and colon) in names for users and / or groups. Characters are used in file names exactly as they are read from the archive.

10.1.3 Multiple Volumes

archive/interchange file format
P. The format-reading utility shall, in an *implementation-defined* manner, determine what file to read as the next file.

S. The format reading utility opens `/dev/tty` and prompts the user for the next volume when an EOF is encountered.

POSIX.1 Section 11, Synchronization

11.2.3.2 Initialize/Open a Named Semaphore: Description

P. If *name* does not begin with the slash character, the effect is *implementation-defined*. The interpretation of slash characters other than the leading slash character in `name` is *implementation-defined*.

S. The Semaphore option is not supplied in Solaris 2.5; hence, there is no implementation-specific behavior.

11.3.1.2 Mutex Initialization Attributes: Description

P. Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are *implementation-defined*.

S. There are no additional attributes in Solaris 2.5.

11.4.1.2 Condition Variable Initialization Attributes: Description

P. Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are *implementation-defined*.

S. There are no additional attributes in Solaris 2.5.

POSIX.1 Section 12, Memory Management

P. Memory locking guarantees the residence of portions of the address space. It is *implementation-defined* whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses.

S. Memory-locking does not lock in translation.

12.1.1.2 Lock/Unlock a Process’s Address Space: Description

P. If MCL_FUTURE is specified, and the automatic locking of future mappings eventually causes the amount of locked memory to exceed the amount of available physical memory or any other *implementation-defined* limit, the behavior is *implementation-defined*. The manner in which the implementation informs the application of these situations is *implementation-defined*.

S. The *mmap()* function will return an error code of [EAGAIN] when resources do not permit the memory to be locked. If the mapping is an attempt to grow the stack, a SIGSEGV signal is sent to the process.

12.1.1.4 Lock/Unlock a Process’s Address Space: Errors

P. For each of the following conditions, if the condition is detected, the *mlockall()* function shall return -1 and set *errno* to the corresponding value:

[ENOMEM]	Locking all of the pages currently mapped into the process’s address space would exceed an *implementation-defined* limit on the amount of memory that the process may lock.

S. There is no per-process limit on how much memory may be locked.

12.1.2.4 Lock/Unlock a Range of Process Address Space: Errors

P. For each of the following conditions, if the condition is detected, the *mlock()* function shall return -1 and set *errno* to the corresponding value:

[ENOMEM]	Locking the pages currently mapped by the specified range would exceed an *implementation-defined* limit on the amount of memory that the process may lock.

S. There is no per-process limit on how much memory may be locked.

12.2.1.2 Map Process Addresses to a Memory Object: Description

P. MAP_FIXED informs the system that the value of pa shall be *addr* exactly. It is *implementation-defined* whether MAP_FIXED is supported.

S. Solaris 2.5 supports MAP_FIXED; however, its use is discouraged.

P. When MAP_FIXED is not set, the system uses `addr` in an *implementation-defined* manner to arrive at pa.

S. The value of *addr* is ignored. An unmapped part of the address is allocated.

12.3.1.2 Open a Shared Memory Object: Description

P. If *name* does not begin with the slash character, the effect is *implementation-defined*. The interpretation of slash characters other than the leading slash character in `name` is *implementation-defined*.

S. The Shared Memory Objects option is not supported in Solaris 2.5; hence, there is no implementation-specific behavior.

12.4.1.1.1 Process Memory Locking: Models

P. The page size is *implementation-defined* and is available to applications as a compile time symbolic constant or at run-time via sysconf().

S. The page size is dependent on the underlying hardware.

POSIX.1 Section 13, Execution Scheduling

13.2 Scheduling Policies

P. Three scheduling policies are specifically required; others may be *implementation-defined*.

S. No other policies are defined by the implementation.

13.2.3 Scheduling Policies: SCHED_OTHER

P. Conforming implementations shall include one scheduling policy identified as SCHED_OTHER (which may execute identically with either the FIFO or round robin scheduling policy). *Conforming implementations shall document* the behavior of this policy as described in the definition of scheduling policy. The effect of scheduling processes with the SCHED_OTHER policy in a system in which other processes are executing under SCHED_FIFO or SCHED_RR shall thus be *implementation-defined.*

S. The Priority Scheduling Option is not supported in Solaris 2.5, hence, there is no implementation-specific behavior.

13.3.1.2 Set Scheduling Parameters: Description

P. The conditions under which one process has permission to change another process's scheduling parameters are *implementation-defined.*

S. If the target process has a SCHED_FIFO or SCHED_RR policy, the calling process must have a SCHED_FIFO or SCHED_RR policy or must have a UID of zero. If the target process has a SCHED_OTHER policy, the calling process must have either the same effective and real user ID as the target process's real or saved user ID or must have a UID of zero.

P. If the current scheduling policy for the process specified by pid is not SCHED_FIFO or SCHED_RR, including SCHED_OTHER, the result is *implementation-defined.*

S. If the target process has the SCHED_OTHER policy, the time-sharing scheduling parameters are set for the target process from the sched_param structure.

13.3.3.2 Set Scheduling Policy and Scheduling Parameters: Description

P. The conditions under which one process has the appropriate privilege to change another process's scheduling parameters are *implementation-defined.*

S. If the target process has a SCHED_FIFO or SCHED_RR policy, the calling process must have a SCHED_FIFO or SCHED_RR policy or must have a UID of zero. If the target process has a SCHED_OTHER policy, the calling process must have either the same effective and real user ID as the target process's real or saved user ID or must have a UID of zero.

P. Implementations may require that the requesting process have permission to set its own scheduling parameters or those of another process. Additionally, *implementation-defined* restrictions may apply as to the appropriate privileges required to set a process's own scheduling policy, or another process's scheduling policy, to a particular value.

S. If the target process has the SCHED_OTHER policy, the time-sharing scheduling parameters are set for the target process from the sched_param structure**.**

13.4.1 Thread Scheduling Attributes

P. The default scheduling contention scope value is *implementation-defined*. The default values of other scheduling attributes are *implementation-defined.*

S. The default value of the scheduling contention scope value is PTHREAD_SCOPE_PROCESS. The default values of other scheduling attributes are shown in table 7-1.

Table 8-1 Scheduling Attributes Default Values

Attribute	**Default**	**Default Value**
contentionscope	Resource competition within process	PTHREAD_SCOPE_PROCESS
detachstate	Joinable by other threads	PTHREAD_CREATE_JOIN
stackaddr	Allocated by system	NULL
stacksize	1 megabyte	NULL
priority	Parent (calling) thread's priority	NULL
policy	Determined by system	SCHED_OTHER
inheritsched	Explicitly defined	PTHREAD_EXPLICIT_SCHED

13.4.2 Scheduling Contention Scope

P. The system scheduling attributes of a thread created with PTHREAD_SCOPE_PROCESS scheduling contention scope are the *implementation-defined* mapping into system attribute speace of the scheduling attributes with which the thread was created.

S. For Solaris, a thread with the PTHREAD_SCOPE_PROCESS scheduling contention scope is implemented as a Solaris “unbound” thread. An unbound thread is multiplexed with other unbound threads to run on top of a lightweight process (LWP). The LWP is a kernel recognized entity and its default system scheduling attributes are defined by the “TIME-SHARING CLASS” (see `priocntl(2)`). An unbound thread's system scheduling attributes, once it starts running on an LWP, are identical to the TIME-SHARING CLASS. At this point, a thread's scheduling attributes specified at thread creation time, or subsequently via `pthread_setschedparam()`, are essentially ignored in their mapping to the system scheduling attributes, which are now completely defined by the TIME-SHARING CLASS.

However, when the thread is not running on an LWP (i.e., is sleeping or runnable), the thread's system scheduling behavior is impacted by the priority specified in the attributes used to create this thread or specified via `pthread_setschedparam()`.

For runnable unbound threads, the highest priority runnable thread will be dispatched to run on an LWP. A lower priority thread cannot be dispatched to run on an LWP, while a higher priority thread is runnable and on the run queue.

For sleeping unbound threads, a wake-up issued on a synchronization object will wake up the highest priority thread sleeping for that synchronization object. Once a thread is dispatched on an LWP, its system scheduling behavior is, as specified above, defined by the LWP's scheduling attributes.

13.4.3 Scheduling Allocation Domain

P. The choice of scheduling allocation domain size and the level of application control over scheduling allocation domains shall be *implementation-defined.* Conforming implementations may change the size of scheduling allocation domains and the binding of threads to scheduling allocation domains at any time.

S. In Solaris 2.5, by default, a thread can run on any of the processors in the system. Hence, the default scheduling allocation domain size is the number of processors online in the system. This value may be obtained via a call to `sysconf(_SC_NPROCESSORS_ONLN)`. This value may be changed via calls to `p_online(2)` by an application with root privileges. This would affect applications' scheduling allocation domain. An application may also change its scheduling allocation domain via calls to `processor_bind(2)`. In future releases of Solaris, exclusive binding of processors may be available. This implies that applications with the appropriate privileges may affect other applications' scheduling allocation domain if they specify exclusive binding of processors.

P. For application threads with scheduling allocation domains of size greater than one, the rules defined for SCHED_FIFO and SCHED_RR in 13.2 shall be used in an *implementation-defined* manner. Each thread with system scheduling contention scope competes for the processors in its scheduling allocation domain in an *implementation-defined* manner according to its priority.

S. The scheduling policies SCHED_FIFO and SCHED_RR are optional policies under POSIX, i.e., a POSIX conformant implementation need not support these policies. Solaris 2.5 does not support these two policies and so this section is not relevant. Of course, the semantics of SCHED_FIFO and SCHED_RR may be obtained via equivalent Solaris interfaces (see `priocntl(2)`).

13.5.2.2 Dynamic Thread Scheduling Parameters Access: Description

P. For SCHED_OTHER, the affected scheduling parameters are *implementation-define.*

S. For SCHED_OTHER, there is only one affected scheduling parameter: that is the priority specified in the `sched_param` structure. See 13.4.2 for information on the Solaris definition for the impact of this priority on threads in the PTHREAD_SCOPE_PROCESS contention scope.

POSIX.1 Section 14, Clocks and Timers

14.2.1.2 Clock and Timer Functions: Description

P. The resolution of any clock can be obtained by calling *clock_getres()*. Clock resolutions are *implementation-defined* and are not settable by a process.

S. The clock resolution depends on the underlying hardware.

P. The effect of setting a clock via *clock_settime()* on armed pre-process timers associated with that clock is *implementation-defined.*

S. The timer expires at the same moment it would have expired had the clock not been changed.

P. The appropriate privilege to set a particular clock is *implementation-defined.*

S. Only a process with UID zero can set the clock CLOCK_REALTIME.

14.2.2.2 Create a Per-Process Timer: Description

P. The behavior for any other value of *sigev_notify* is *implementation-defined.*

S. No other value of *sigev_notify* is supported.

P. If clock_id specifies the CLOCK_REALTIME system clock, then the default signal, when evp is NULL shall be SIGALRM. For any other clock, the default signal number is *implementation-defined.*

S. No other clocks are supported.

14.2.4.2 Per-Process Timers: Description

P. The overrun count returned shall contain the number of extra timer expirations which occurred between the time the signal was generated (queued) and when it was delivered, up to but not including an *implementation-defined* maximum of {DELAYTIMER_MAX}.

S. The maximum overrun count is INT_MAX.

POSIX.1 Section 15, Message Passing

15.1.1. Data Definitions for Message Queues: Data Structures

P. The header `<mqueue.h>` shall define the following *implementation-defined* types:

mqd_t	Used for message queue descriptors

S. The type mqd_t is declared:

```
typedef     void     *mqd_t;
```

P. The header `<mqueue.h>` defines the following *implementation-defined* structures:

struct sigevent	As specified in 3.3.1

S. struct sigevent {

```
        int             sigev_notify;      /*notification mode */
        int             sigev_signo;       /*signal number */
        union sigval    sigev_value;       /* signal value */
};
```

15.2.1.2 Open a Message Queue: Description

P. The interpretation of slash characters other than the leading slash character in `name` is *implementation-defined.*

S. The Message Passing Option is not supported in Solaris 2.5; hence, there is no implementation-specific behavior.

O_CREAT
P. The “file permission bits” shall be set to the value of mode. When bits in mode other than file permission bits are set, the effect is *implementation-defined*.

S. The Message Passing Option is not supported in Solaris 2.5; hence, there is no implementation-specific behavior.

P. If *attr* is `NULL`, the message queue is created with *implementation-defined* default message queue attributes.

S. The Message Passing Option is not supported in Solaris 2.5; hence, there is no implementation-specific behavior

POSIX.2 9

Portable Operating System Interface –Part 2: Shell and Utilities (POSIX.2)

In 1992, IEEE Standard 1003.2–1992 became part of the POSIX series of standards. Referred to as “POSIX.2,” IEEE Standard 1003.2–1992 defines a standard interface and environment for applications that require a shell command language interpreter and a set of common utility programs. POSIX.2 is complimentary to ISO / IEC 9945–1:1990.

IEEE Std 1003.2-1992 also supplements the application portability interfaces to promote the “portability” of users and programmers between conforming systems. The User Portability Utilities Option extends the list of utilities, and features of utilities used primarily for application portability, to provide a common interactive environment.

This standard, referred to as “POSIX.2,” is based upon documentation and the knowledge of existing programs that assume an interface and architecture similar to that described by POSIX.1. Within this chapter the POSIX.2 standard is referred to in places as “the standard.”

Scope

To comply with section 1.3.1.2 (*Documentation*), this document describes the behavior of features in the SunOS 5.5 operating system which are described in the standard as *implementation-defined* or for which it is stated that implementations may vary. It does not describe any extensions or enhancements outside the scope of the standard.

Note – Section 2.2.1.5 of the standard defines the term *implementation-defined* as follows: “An indication that the implementation provider shall define and document the requirements for correct program constructs and correct data of a value or behavior.”

The information contained within this chapter does not replace the POSIX.2 standard; rather, it serves as an adjunct to the standard for supplying the technical information needed by application developers to write source code within the SunOS 5.5 operating system framework.

Audience

This explication is for the experienced C programmer who, when writing an applications program designed to conform to the standard, needs to know the specific behavior of the *implementation-defined* features mentioned in the standard.

Each subsection is prefaced by the appropriate section taken directly from the standard and has the corresponding section number attached to the title.

Note – For maximum portability, applications should not depend upon any particular behavior that is *implementation-defined*.

Notation Used in the Remainder of this Chapter

The following format is used to identify which passage of text is quoted from the standard and which passage of text describes how the SunOS 5.5 operating system implements that area.

- **P.** stands for POSIX.
- **S.** stands for SunOS 5.5.

Section numbers cited in the remainder of this chapter correspond to those of the standard. When creating an application program, this format will help you to quickly locate additional information that you need from the standard.

Implementation-defined Areas of POSIX.2

POSIX.2 Section 1, General

1.3.1 Implementation Conformance

1.3.1.1 Requirements
P. The system may provide additional or enhanced utilities, functions, or facilities not required by this standard. Nonstandard extensions should be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of utilities, functions, or facilities defined by this standard.

S. The SunOS 5.5 reference manuals describe the utilities, functions, and facilities provided by the SunOS 5.5 operating system.

P. The conformance document shall define an execution environment in which an application can be run with the behavior specified by the standard.

S. When Solaris 2.5 is installed with SPARCompiler 4.0 on a SPARC based platform, or with ProCompiler™ C 3.0 and patch 102486-02 on an x86 based platform, and the application searches for standard utilities in the directories specified by the output of the `getconf PATH` command in the order specified from beginning to end, the utilities will behave as specified by POSIX.2.

1.3.1.2 Documentation
P. The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies.

S. IEEE Std 1003.2-1992 and IEEE Std 1003.2a-1992 are supported.

P. The conformance document may also list software standards approved by ISO/IEC or any ISO/IEC member body that are available for use by a Conforming POSIX.2 Application.

S. The available software depends on which packages have been installed.

POSIX.2 Section 2, Terminology and General Requirements

2.2.2 General Terms

2.2.2.8: appropriate privileges

P. An *implementation-defined* means of associating privileges with a process with regard to the function calls and function-call options defined in POSIX.1 that need special privileges. There may be zero or more such means.

S. The following is a list of references to “appropriate privileges” in the POSIX.2 standard:

3.5.3 Variables See section 3.5.3 in this document.

4.7.2 Description {of `chmod`} See POSIX.1 section 5.6.4.2 *Description {of Change File Modes*}.

4.30.2 Description {of `id`} See section 4.30.2 *Description {of* `id`} in this document.

4.35.2 Description {of `localedef`} See section 4.35.2 *Description {of* `localedef`*}* in this document.

4.35.4 Operands {of `localedef`} See section 4.35.4 *Operands* {*of* `localedef`} See section 4.35.4 Operands {of localedef} in this document.

5.17.2 Description {of `mesg`} See section *5.17.2 Description* {*of* `mesg`} in this document.

5.20.2 Description {of `nice`} See section 5.20.2 *Description* {*of* `nice`} in this document.

5.20.3 Options {of `nice`} See section 5.20.3 *Options* {*of* `nice`} in this document.

5.24.2 Description {of `renice`} See section 5.24.2 *Description* {*of* `renice`} in this document.

5.24.3 Options {of `renice`} See section *5.24.3 Options* {*of* `renice`} in this document.

5.28.2 Description {of `talk`} See section 5.28.2 *Description* {*of* `talk`} in this document.

5.37.2 Description {of `write`} See section 5.37.2 *Description* {*of* `write`} in this document.

2.2.2.27: byte
P. A byte is composed of a contiguous sequence of bits, the number of which is *implementation-defined.*

S. On Solaris 2.5, a byte is composed of 8 bits.

2.2.2.61: extended security controls
P. The access control and privilege mechanisms have been defined to allow *implementation-defined* extended security controls.

S. Access Control Lists are drfined in *SunOS Reference Manual* in the getfacl(1) man page.

2.2.2.65: file
P. A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, and directory. Other types of files may be defined by the implementation.

S. Symbolic links are also defined.

2.2.2.68: file group class
P. A process is in the file group class of a file if the process is not in the file owner class and if the effective group ID or one of the supplementary group IDs of the process matches the group ID associated with the file. Other members of the class may be *implementation-defined.*

S. There are no additional file class members.

2.2.2.93: job control
P. POSIX.1 conforming implementations may optionally support job control facilities.

S. Job control is supported

2.2.2.120: parent process ID
P. After the lifetime of the creator has ended, the parent process ID is the process ID of an *implementation-defined* system process.

S. If a child process continues to exist after its creator process ceases to exist, the child process is inherited by `init`. The `init` process ID is 1.

2.2.2.121: pathname
P. A pathname that begins with two successive slashes may be interpreted in an *implementation-defined* manner.

S. Same as one slash.

2.2.2.141: read only file system
P. A file system that has *implementation-defined* characteristics restricting modifications.

S. A read-only file system does not allow for modification of its files or directories.

2.4 Character Set

P. Use of a locking-shift encoding with any of the standard utilities or the optional C-language functions that describe character (versus byte) or text-file manipulation is *implementation-defined.*

S. Use of locking-shift encodings is not supported.

2.4.1 Character Set Description File

P. It is *implementation-defined* whether or not users or applications can provide additional character set description files.

S. Additional character set description files are allowed.

P. Implementations supporting other byte sizes may allow constants to represent values larger than those that can be represented in 8-bit bytes, and to allow additional digits in constants. The manner in which these constants are represented in the character stored in the system is *implementation-defined.*

S. Bit sizes other than 8 bits are not supported.

2.5 Locale

P. Locales other than those supplied by the implementation can be created via the `localedef` utility provided that the {POSIX2_LOCALEDEF} symbol is defined on the system. Otherwise only the implementation provided locales(s) can be used.

S. The `localedef` utility is provided. Other locales can be created using this utility.

P. When the value of a locale environment variable begins with a slash (`/`), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, etc.) used to store the locale definition is *implementation-defined*. If the locale value does not begin with a slash, the mechanism used to locate the locale is *implementation-defined*.

S. The mechanism to locate the locale is defined by the implementation of the `setlocale()` system call provided by Solaris 2.5. On Solaris 2.5, `setlocale()` supports composite locale names.

2.5.2: Locale Definition

P. If the file contains source definitions for more than one category, *implementation-defined* categories, if present, shall appear after the categories defined by this clause.

S. No additional categories or any additional keywords in any locale category source definition are defined.

P. (2) A character can be represented by the character itself, in which case the value of the character is *implementation-defined*.

S. All three supported locales, C, POSIX, and en_US, use the ISO8859-1 character encoding values.

2.5.2.1 LC_CTYPE

P. When the implementation automatically includes a missing character, it shall have an encoded value dependent on the *charmap* file in effect; otherwise, it shall have a value derived from an *implementation-defined* character mapping.

S. The character mapping used is the ISO8859-1 codeset.

2.5.2.5 LC_TIME

P. It is *implementation-defined* whether the following optional keywords shall be recognized.

era
era_year
era_d_fmt
alt_digits

S. These keywords are recognized.

2.5.3 Locale Definition Grammar

P. Any grammars for additional categories and keywords are *implementation-defined*.

S. Under the LC_CTYPE category we provide the `cswidth` keyword that specifies the size in bytes of the codesets. The Solaris internal prepresentation uses EUC which has four codesets.

The syntax of `cswidth` is:

```
cswidth     <n1>:<d1>, <n2>:<d2>, <n3>:<d3>
```

where <n?> is the number of byt4es for the codeset and <d?> is the number of display positions for the codeset for codesets 1, 2, and 3 respectively. Codeset 0 is always 1 byte.

LC_TIME recognizes the `date_fmt` keyword.

The syntax is:

```
date_fmt    "date format"
```

This value is accessed when a `%C` is sent to the date/time code.

2.6 Environment Variables

LANG
LC_COLLATE
LC_TYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME

P. Additional semantics of (these) variable(s), if any, are *implementation-defined*.

S. *The POSIX1003.2 Shell and Utility Application Interface* defines no additional semantics of these environment variables.

PATH
P. If **PATH** is unset or is set to null, the path search is *implementation-defined*.

S. When `PATH` is not set, Solaris supplies the default path `/usr/bin`.

P. If the **LANG** variable is not set or is set to the empty string, the *implementation-defined* default locale shall be used.

S. On Solaris 2.5, the implementation-defined default locale is “C”.

P. If **LANG** (or any of the **LC_*** environment variables) contains one of a set of *implementation-defined* values, the standard utilities shall behave in accordance with the rules in a corresponding *implementation-defined* locale description for the associated category.

S. This is true for all locales that Sun supports.

P. Additional criteria for determining a valid locale name are *implementation-defined*.

S. None.

2.9.1.5 File Removal

P. When a directory that is the root directory or current working directory of any process is removed, the effect is *implementation-defined*.

S. If the directory indicated in the call to `rmdir()` is a mount point for a mounted filesystem, `rmdir()` sets `errno` to [EBUSY] and returns -1.

2.11.5.2 Input Files

P. *Implementations shall define* in the conformance documentation those utilities that are limited by constraints other than file system space, available memory, and other limits specifically cited by this standard; and identify what the constraint is; and indicate a way of estimating when the constraint would be reached.

S. The `cu`, `uucp`, `uux`, `uuname`, `uulog`, `uupick`, and `uuto` configuration files include:

```
/etc/uucp/Config   /etc/uucp/Dialers  /etc/uucp/Poll
/etc/uucp/Devconfig/etc/uucp/Grades   /etc/uucp/Sysfiles
/etc/uucp/Devices  /etc/uucp/Limits   /etc/uucp/Systems
/etc/uucp/Dialcodes/etc/uucp/Permissions
```

These file must be edited or otherwise manipulated in the POSIX locale. The behavior is undefined if multibyte characters are present in the configuration files.

P. Similarly, some utilities descend the directory tree (recursively). Implementations also shall document any limits that they may have in descending the directory tree that are beyond the limits cited by this standard.

S. Except for `du`, the *POSIX 1003.2 Shell and Utility Application Interface* meets the performance limits of the POSIX.2 standard. The depth of the hierarchy which can be properly processed by `du` is limited by the number of files which a single process may have open at once.

2.14 Terminal Characteristics

P. The implementation shall document which terminal types it supports and which of these features and utilities are not supported by each terminal. This implementation-defined list of terminals

-Shall include at least one terminal type that is capable of supporting all of the standard utilities and all of their features, if the {POSIX2_CHAR_TERM} option is provided.

-May group terminals in terms of families or equivalences to other documented terminal types.

-Need not consist of an exhaustive list of terminal models when the implementor considers that some terminal types are used too infrequently to be listed.

S. Any terminal for which a user can obtain a TERMINFO description and run through the `tic` utility is capable of supporting all POSIX.2 utilities.

POSIX.2 Section 3, Shell Command Language

3.5.3 Variables

PS1
P. For users who have additional *implementation-defined* privileges, the default may be another, *implementation-defined* value.

S. If a user's effective ID has the value of zero (0), the default value of **PS1** changes from "$ " to "# ".

3.7 Redirection

P. In this standard, open files are represented by decimal numbers starting with zero. It is *implementation-defined* what the largest value can be; however, all implementations shall support at least 0 through 9 for use by the application.

S. The largest value that can be used as a file descriptor is 9. This means that the user can have up to ten files open, including the standard input, standard output, and standard error files.

POSIX.2 section 4, Execution Environment Utilities

4.1.7.3 Variables and Special Variables

`ENVIRON`
P. In all cases where the behavior of `awk` is affected by environment variables...the environment used shall be the environment at the time `awk` began executing; it is *implementation-defined* whether any modification of `ENVIRON` affects this environment.

S. Changing `ENVIRON` has no effect.

`SUBSEP`
P. The subscript separator string for multidimensional arrays; the default value is *implementation-defined.*

S. The `awk` utility uses the value \034 (the ASCII FS control character).

4.1.7.6.2.3 Input/Output and General Functions

close
P. The limit on the number of open expression arguments is *implementation-defined.*

S. The `awk` utility defines this limit as the value {OPEN_MAX}-4. The minimum value for this limit is 12 and by default, this value is 60.

4.1.7.8 `awk` Lexical Conventions

Table 4-2 \ddd
P. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is *implementation-defined.*

S. The `awk` utility supports 8 bit bytes.

4.2.2 `basename` – Return Nondirectory Portion of Pathname: Description

P. If *string* is `//`, it is *implementation-defined* whether steps (2) through (5) are skipped or processed.

S. "`/`" is returned.

4.5.2 `cd` – Change Working Directory: Description

P. If **HOME** is empty or is undefined, the default behavior is *implementation-defined.*

S. In both cases the `cd` utility gives the following error message:

```
ksh:cd. bad directory
```

4.5.4 Operands

directory
P. If directory is –, the results are *implementation-defined.*

S. The `cd` utility handles this case by performing the equivalent of the command `"cd "$OLDPWD" && pwd"`.

4.7.2 `chmod` – Change File Modes: Description

P. It is *implementation-defined* whether and how the `chmod` utility affects any alternate or additional file access control mechanism being used for the specified file.

S. The `chmod` utility does not effect access control lists.

4.7.7 `chmod` – Extended Description

P. When using the symbolic mode form on a regular file, it is *implementation-defined* whether or not:

(1) Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all execute bits are currently clear and none are being set are ignored,

(2) Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-on-execution bits, or

(3) Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bit when all execute bits are currently clear are ignored.

S. (1) `{u|g|a}{+|=}s` are ignored. `{u|g|o|a}{+|=}l` are allowed. The letter `l` is the file-lock bit (equivalent to the S_GID bit).

(2) ID bits are also cleared.

(3) Requests are not ignored. Note: The S_GID bit must be referred to as the letter `l`.

P. When using the symbolic mode form on other file types, it is *implementation-defined* whether or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bit are honored.

S. In the case of set, the requests are honored when the execution bits are set or being set. In the case of clear, the requests are honored. Note: The S_GID bit must be referred to as the letter `l`.

P. For each bit set in the octal number, the corresponding file permission bit shall be set; all other file permission bits shall be cleared. For regular files, each bit set in the octal number corresponding to the set-user-ID-on-execution or set-group-ID-on-execution bits shall be set; if these bits are not set in the octal number, they shall be cleared. For other file types, it is *implementation-defined* whether or not request to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are honored.

S. The requests are honored.

4.13.2 `cp` – Copy Files: Description

(2) (c)
P. If *dest_file* exists and it is a file type not specified by POSIX.1, the behavior is *implementation-defined.*

S. The behavior in this case depends upon the underlying behavior of the `stat()` system function when applied with the *dest_file* name. For instance, if *dest_file* was a symbolic link to a directory, `stat()` returns information indicating that *dest_file* was a directory and the behavior will be as if *dest_file* was a directory. For all other non-directory types, the behavior would be the same as (2)(d).

(4) (a)
P. If the `-r` option was specified, the behavior is *implementation-defined.*

S. The `cp` utility defines the behavior of the `-r` option as being similar to the behavior of the `-R` option except when copying SPECIAL files. The `-r` option actually tries to read SPECIAL files, while `-R` recreates them. For example, if the `-R` option is specified and the source file is of type FIFO, then the destination will be another file of type FIFO. If the `-r` option is specified instead, then the destination file will be a regular file consisting of the contents of the FIFO source file.

(4)(b)(2)
P. If *source_file* is a file of type FIFO, the file permission bits shall be the same as those of *source_file*, modified by the file creation mask of the user if the `-p` option was not specified. Otherwise, the permissions, owner ID, and group ID of *dest_file* are *implementation-defined.*

S. The `cp` utility sets the *dest_file* privileges to those of the *source_file* and the owner ID and group ID to the current effective user and group IDs.

P. If the implementation provides additional or alternate access control mechanisms, their effect on copies of files is *implementation-defined.*

S. The `cp` utility defines no alternate or additional file access control mechanisms.

4.13.3 Options

`-p`
P. (3) Other *implementation-defined* bits may be duplicated as well.

S. The `cp` utility duplicates all of the bits of *st_mode* member from the `stat()` function. In particular, on systems supporting the UNIX variable *S_ISVTX*, `cp` duplicates the "sticky bit".

`-r`
P. The treatment of special files is *implementation-defined*.

S. With `-r` the `cp` utility attempts to open the special file, and copy its contents. For example, if the `-R` option is specified, and the source file is of type FIFO, the destination is another file of type FIFO. If, instead, the `-r` option is specified, the destination file will be a regular file, consisting of the contents of the FIFO.

4.18.2 `dirname` – Return Directory Portion of Pathname: Description

P. (6) If the remaining *string* is `//`, it is *implementation-defined* whether steps (7) and (8) are skipped or processed.

S. The `dirname` utility processes steps (7) and (8) when converting a string of `//` to a filename; that is, `dirname //` converts to `/`.

4.19.4 Operands

string
P. If the first operand is "`-n`" or if any of the operands contain a backslash (\) character, the results are *implementation-defined*.

S. The `echo` utility takes no special action for "`-n`"; the operand is echoed directly.

The historical SVID functionality as an extension to the standard is supported and includes the following escape sequences in the `echo` operands:

\a	Write an `<alert>` character.
\b	Write a `<backspace>` character.
\c	Suppress the `<newline>` character that otherwise follows the final argument in the output, with everything after `\c` in input being ignored.
\f	Write a `<form-feed>` character.
\n	Write a `<newline>` character.
\r	Write a `<carriage-return>` character.
\t	Write a `<tab>` character.
\v	Write a `<vertical tab>` character.
\\	Write a backslash character.
\0*num*	Write an 8 bit value that is the 0-, 1-, 2-, or 3 digit octal number *num*.
\X	When X is not one of the preceding characters, the `echo` utility simply echoes the two-character sequence \X.

4.20.7.3.14 List Command

P. If the size of a byte on the system is greater than nine bits, the format used for nonprintable characters is *implementation-defined*.

S. The `ed` utility does not support byte sizes other than 8 bits.

4.24.4 Operands

`exec`
P. If a utility_name or argument string contains the two characters `{}`, but not just the two characters `{}`, it is *implementation-defined* whether `find` replaces those two characters with the current pathname or uses the string without change.

S. The `find` utility uses the string without change.

4.30.2 `id` – Return User Identity: Description

P. If a user operand is provided and the process has the appropriate privileges, the user and group IDs of the selected user shall be written.

S. In this case, all processes are considered to have "appropriate privileges."

4.33.2 `ln` – Link Files: Description

P. If the last operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is *implementation-defined*.

S. If the file type is not specified by POSIX.1, it is treated as a non-directory file.

4.33.4 Operands

source_file
P. A pathname of a file to be linked. This can be a regular or special file; whether a directory can be linked is *implementation-defined*.

S. Links to directories are not supported. Symbolic links, however, to directories are supported.

4.34.3 Options

`-a`
P. The manner in which the implementation determines what other locales are available is *implementation-defined*.

S. The `locale` utility lists each directory in `/usr/lib/locale` when the `-a` option is specified.

4.34.4 Operands

`name`
P. It is *implementation-defined* whether any keyword values are written for the categories **LC_CTYPE** and **LC_COLLATE**.

S. Keyword values are written for **LC_CTYPE**, but not for **LC_COLLATE**.

4.35.2 `localedef` – Define Locale Environment: Description

P. It is *implementation-defined* whether users shall have the capability to create new locales, in addition to those supplied by the implementation. If the symbolic constant {POSIX2_LOCALEDEF} is defined, then the system supports the creation of new locales.

S. Users are permitted to create locales using `localedef`.

4.36.2 `logger` – Log Messages: Description

`-name`
P. It is *implementation-defined* whether messages written in locales other than the POSIX Locale are effective.

S. Messages written in other locales are permitted. The `logger` utility always executes in the POSIX locale and treats the message operand as a sequence of bytes.

4.39.3 Options

`-a`
P. Entries beginning with a period (.) shall not be written out unless they are explicitly referenced, the `-a` option is supplied, or an *implementation-defined* condition causes them to be written.

S. The `ls` utility provides other options, namely `-f` and `-A`, which also enable the listing of entries beginning with a period(.). There are no other conditions that would allow entries beginning with a period (.) to be listed.

4.39.5.3 Environment Variables

COLUMNS
P. If **COLUMNS** is not set or is invalid, an *implementation-defined* number of column positions shall be assumed, based on the knowledge of the output device by the implementation.

S. The `ls` utility obtains the numbers of columns. The approach used to determine the numbers of columns is as follows:

- If the **COLUMNS** environment variable is a valid integer number, then it is used;
- If the **COLUMNS** variable is not a valid integer, use the information returned by the `ioctl(STDOUT_FILENO, TIOCGWINSZ, $wininfo)` system function.
- If the values remain invalid, the value of 80 is used for columns.

4.39.6.1 Standard Output

P. The default format shall be to list one entry per line to standard output. If the output is to a terminal, the format is *implementation-defined*.

S. The ls utility uses a multi-column format, as if the user specified `-C`.

P. If the file is a character special or block special file, the size of the file may be replaced with *implementation-defined* information associated with the device in question.

S. The `ls` utility replaces the file size with the numeric values of the major and minor device numbers displayed with the format “`%u, %u`”. The major and minor device numbers are obtained by taking the *st_rdev* structure member returned from the `stat()` system function and passing it to the `major()` and `minor()` system functions. See `stat(2)` and `mkdev(3)` manpages for further explanations.

P. Implementations may add other characters to this list [of *entry type* characters] to represent other, *implementation-defined*, file types.

S. The `ls` utility also uses the character `x` to indicate “none of the above” file types.

The character `s` is used to indicate “sockets”.
The character `l` is used to indicate symbolic “links”.
The character `L` is used to indicate mandatory locking.

P. Implementations may add other characters to this list for the third character position.

S. The `ls` utility uses the characters `T` or `t` to indicate the “sticky bit” file attribute. This bit is defined as the `S_ISVTX` bit mask defined in `<stat.h>`.

4.40.6.3 Output Files

P. When a message from the system mailbox or entered by the user is not a text file, it is *implementation-defined* how such a message is stored in files written by `mailx`.

S. The mailx utility attempts to process the file and therefore produces inappropriate results.

4.40.7.1 `mailx` Internal Variables

`crt=`*number*

P. Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable. The default shall be `nocrt`. If it is set to null, the value used is *implementation-defined.*

S. If it is set to null, the value is whatever the screen size is minus 2.

4.40.7.3 `mailx` Command Escapes

`~h`

P. If standard input is a terminal, prompt for a Subject line and the To, Cc, and Bcc lists. Other *implementation-defined* headers may also be presented for editing.

S. None.

4.43.2 `mv` – Move Files: Description

P. If any operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is *implementation-defined.*

S. For symbolic link file types, the behavior of `mv` is to reference the symbolic link file itself when validating the existence of the source file arguments (e.g. uses `lstat()`) and to reference the file to which the symbolic link points when validating and referencing the target argument (e.g. uses `stat()`).

4.45.7 Extended Description

P. The default number of bytes transformed by output type specifiers `d`, `f`, `o`, `u`, and `x` shall correspond to the various C-language types as follows. If the `c89` compiler is present on the system, these specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes are *implementation-defined*. The POSIX.2 standards expands on this with the explanation that “For the type specifier characters `d`, `o`, `u`, and `x`, the default number of bytes shall correspond to the size of the underlying implementation’s basic integral data type...”

S. The default number of bytes transformed by the type specifier characters `d`, `o`, `u`, and `x` is 4 bytes (e.g. based on the C-language type `int`).

P. For these specifier characters [`d`, `o`, `u` and `x`], the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the c-language types `char`, `short`, `int`, and `long`.

S. The optional number of bytes to be converted by the characters C, S, I, and L are 1, 2, 4, and 4 respectively on both SPARC and Intel.

P. For the type specifier character `f`, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating point date type.

S. The default number of bytes used for the type specifier `f` is 8 bytes (e.g. based on the C-language type `double`).

P. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types `float`, `double` and `long double`.

S. The optional number of bytes to be converted for the characters F, D, and L are 4, 8, and 8 respectively on both SPARC and Intel.

P. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

S. For SPARC, the most significant byte is stored in the lower memory address. For Intel, the least significant byte is stored in the lower memory address.

P. If the size of a byte on the system is greater than nine bits, the format used for nonprintable characters is *implementation-defined.*

S. On Solaris 2.5, the size of a byte is 8 bits.

P. When either the `-j` *skip* or `-N` *count* option is specified along with the `c` type specifier, and this results in an attempt to start or finish in the middle of a multibyte character, the result is *implementation-defined.*

S. When the output starts or finishes in the middle of a multibyte character, the partial multibyte character is displayed as single-byte characters.

4.48.2 `pax` – Portable Archive Interchange: Description

copy
P. If the destination directory is a file of a type not defined by POSIX.1 {8}, the results are *implementation-defined*.

S. Copying will cause an error.

P. The default output archive format shall be *implementation-defined*.

S. *ustar* is the default archive format.

P. The `pax` utility shall determine, in an *implementation-defined* manner, what file to read or write as the next file.

S. `Go` is prompted to continue and `Quit` is prompted to exit while reading the same archive file.

4.48.3 Options

`-a`
P. It is *implementation-defined* which devices on the system support appending.

S. Regular disk files and 4 mm tape drives support appending.

`-p` *string*
P. The string shall consist of the specification characters `a`, `e`, `m`, `o`, and `p`, and / or other *implementation-defined* characters.

S. There are no other file characteristics.

Specification character `e`
P. Preserve the user ID, group ID, file mode bits, access time, modification time, and any other *implementation-defined* file characteristics.

S. There are no other characters.

Specification character `p`
P. Preserve the file mode bits. Other, *implementation-defined* file-mode attributes may be preserved.

S. The implementation also provides a bit identified by S_ISVTX. For a directory, this bit determines whether or not an unprivileged user may delete or rename another user’s files from that directory (refer to `chmod(2)` for other files types).

`-x` *format*
P. *Implementation-defined* formats shall specify a default block size as well as any other block sizes supported for character special archive files.

S. There are no *implementation-defined* formats.

4.48.5.2 Input Files

P. The input file named by the archive option-argument, or standard input when the archive is read from there, shall be a file formatted according to one of the specifications in section 10.1 of POSIX.1 {8}, or some other *implementation-defined* format.

S. No other formats are defined.

4.48.6.1 Standard Output

P. In write mode, if `-f` is not specified, the standard output shall be the archive formatted according to one of the specifications in section 10.1 of POSIX.1 {8}, or some other *implementation-defined* format.

S.No other formats are defined.

4.48.6.3 Output Files

P. In write mode, the output file named by the `-f` option argument shall be a file formatted according to one of the specifications in section 10.1 of POSIX.1 {8}, or some other *implementation-defined* format.

S. No other formats are defined.

4.55.7.3 `sed` *Editing Commands*

[2addr]`l`
P. If the size of a byte on the system is greater than nine bits, the format used for non-printable characters is *implementation-defined.*

S. The `sed` utility does not support byte sizes greater than 8 bits.

4.56.5.3 Environment Variables

PS1
P. For users who have specific additional *implementation-defined* privileges, the default may be another *implementation-defined* value.

S. If a user’s effective user ID has the value zero (0), the default value of **PS1** changes from “$” to “#”.

4.59.2 `stty` *– Set the Options for a Terminal: Description*

P. The `stty` utility shall set or report on terminal I/O characteristics for the device that is its standard output. Without options or operands specified, it shall report the settings of certain characteristics, usually those that differ from *implementation-defined* defaults.

S. Certain settings are reported when no options or operands have been specified. The following settings are reported when the characteristic is enabled: `markp`, `spacep`, `oddp`, `evenp`, `cs5`, `cs6`, `cs7`, `cs8`, `cstopb`, `hupcl`, `clocal`, `loblk`, `line`, `rows`, `columns`, `ypixels`, `xpixels`, `min`, `time`, `ignbrk`, `brkint`, `ignpar`, `parmrk`, `inlcr`, `igncr`, `icrnl`, `iuclc`, `ixoff`, `imaxbel`, `olcuc`, `onlcr`, `ocrnl`, `onocr`, `onlret`, `del-fill`, `nul-fill`, `cr`, `nl`, `tab`, `bs`, `vt`, `ff`, `xcase`, `echonl`, `noflsh`, `tostop`, `echoctl`, `echoprt`, `echoke`, `defecho`, `flusho`, `pendin`, `iexten`.

The following settings are reported when the characteristic is disabled: `-parity`, `-cread`, `-inpck`, `-istrip`, `-ixon`, `-ixany`, `-opost`, `-isig`, `-icanon`, `-echo`, `-echoe`, `-echok`.

In addition:

`intr` is reported if it is set to something other than `^c`,
`quit` is reported if it is set to something other than `^l`,
`erase` is reported if it is set to something other than `^?`,
`kill` is reported if it is set to something other than `^u`,
`eof` is reported if it is set to something other than `^d`,
`eol` is reported if it is defined,
`eol2` is reported if it is defined,
`swtch` is reported if it is set to something other than `^z`,

`start` is reported if it is set to something other than `^q`,
`stop` is reported if it is set to something other than `^s`,
`susp` is reported if it is set to something other than `^z`,
`dsusp` is reported if it is set to something other than `^y`,
`rprnt` is reported if it is set to something other than `^r`,
`flush` is reported if it is set to something other than `^o`,
`werase` is reported if it is set to something other than `^w`,
`lnext` is reported if it is set to something other than `^v`,
`speed` is reported when `ispeed` and `ospeed` are identical, otherwise `ispeed` and `ospeed` are reported.

4.59.4.4 Local Modes

`iexten` (`-iexten`)
P. Enable (disable) any *implementation-defined* special control characters not currently controlled by `icanon`, `isig`, `ixon`, or `ixoff`.

S. Special control characters not controlled by `icanon`, `isig`, `ixon`, or `ixoff` are `veolz`, `vswtch`, `vreprint`, `vdiscard`, `vdsusp`, `vwerase`, `vlnext`.

4.59.4.6 Combination Modes

`sane`
P. Reset all modes to some reasonable, unspecified values.

S. `stty sane` is equivalent to:

```
stty cs7 parenb cread -csize -parodd -clocal brkint ignpar \
istrip icrnl ixon imaxbel -ignbrk -parmrk -inpck -inlcr -igncr \
-iuclc -ixoff -ixany isig icanon iexten echo echok echoe echoke\
echoctl -xcase -echonl -noflsh opost onlcr -olcuc -ocrnl -onocr \
-onlret -ofill -ofdel -nldly -crdly -tabdly -bsdly -vtdly -ffdly \
erase ^? kill ^u quit ^l intr ^c eof ^d eol undef
```

4.62.4 Operands

P. Additional *implementation-defined* operators and *primary-operators* may be provided by implementations. The additional implementation-defined operators “(“ and “)” may also be provided by implementations.

S. The `test` utility provides the following additional primaries:

`-a`	True if both *expression1* and *expression2* are true,
`-o`	True if either *expression1* or *expression2* are true,
`(,)`	Allow primaries to be grouped as single expressions, for use with `-a` and `-o` primaries.
`-k`	True if the “sticky bit” is on
`-nt`	True if *file1* is newer than *file2*
`-ot`	True if *file1* is older than *file2*
`-ef`	True if *file1* has the same device and inode as *file2*; that is, same file
`-L`	True if *file* is a symbolic link
`-h`	True if *file* is a symbolic link

4.63.3 Options

P. The range of valid times past the Epoch is *implementation-defined...*

S. The range of valid times past the Epoch depends on the size of the ANSI C arithmetic type `time_t` and the behavior of the ANSI C `mktime()` routine. `time_t` is a signed long, 32 bits. The maximum positive integer is 2147483647, which represents 68 years, 18 days, 3 hours, 14 minutes, and 7 seconds.

From the Epoch (12:00:00 a.m. January 1, 1970), 2147483647 represents the date: Tue Jan 19 03:14:07 GMT 2038.

For the behavior of `mktime()` wee the `mktime(3C)` man page.

4.64.7 Extended Description

\octal
P. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is *implementation-defined.*

S. The `tr` utility does not support byte sizes greater than 8 bits.

4.68.2 `uname` – Return System Name: Description

P. When options are specified, symbols representing one or more system characteristics shall be written to the standard output. The format and contents of the symbols are *implementation-defined.*

S. The `uname` utility supports seven symbols, including the five specified by POSIX.1, section 4.4.1.2 and displays the additional symbols at the end of the output line.

4.68.6.1 Standard Output

P. Additional *implementation-defined* symbols may be written.

S. The additional symbols are machine class and machine processor type. “machine” symbol is called “platform” in `uname` man page.

POSIX.2 Section 5, User Portability Utilities Option

5.2.2 Execute Commands at a Later Time: Description

P. The at-job shall be executed in a separate invocation of the shell, running in a separate process group with no controlling terminal, except that the environment variables, current working directory, file creation mask, and other *implementation-defined* execution-time attributes in effect when the `at` utility is executed shall be retained and used when the at-job is executed.

S. The limits controlled by the `unlimit` facility are also retained, but only applies to `ksh`, `sh`, and `/usr/xpg4/bin/sh`.

5.2.3 Options

`-m`
P. If `-m` is not used, the standard output and standard error of the job shall be provided to the user via an *implementation-defined* mechanism, unless they are redirected elsewhere;

S. stdout and stderr are sent via `/bin/mail`.

`-q` *queuename*
P. By default, at-jobs shall be scheduled in queue “a”. In contrast, queue “b” shall be reserved for batch jobs. The meanings of all other *queuenames* are *implementation-defined.*

S. The `at` utility can use any single-byte character except NULL,'\b', '\n', '\t', and '#' characters as a queuename. All queues have the same functionality as queue 'a', except queue limits and priorities may be set in the `queuedefs` file. See POSIX.2 `queuedefs(4)man` page for more information. 'a' through 'z' are valid queuenames.

5.2.4 Operands

timespec: time
P. The acceptable time-zone names are *implementation-defined.*

S. ZULU, UTC, and GMT (case insensitive)

`%token timezone_name`
P. The name of an optional time-zone suffix to the time field, in an *implementation-defined* manner.

S. ZULU, UTC, and GMT (case insensitive)

5.5.2 `crontab` – Schedule Periodic Background Work: Description

P. If standard output and standard error are not redirected by commands executed from the crontab entry, any generated output or errors shall be mailed, via an *implementation-defined* method, to the user.

S. stdout and stderr are sent via `/bin/mail`.

5.7.4 Extended Description

P. The handling of other files is *implementation-defined.*

S. The `ctags` utility handles C, C++, Pascal, FORTRAN, YACC, and LEX sources.

5.7.7 Operands

P. It is *implementation-defined* what other objects (including duplicate identifiers) produce output.

S. Duplicate identifiers and too many entries produce output.

5.8.6.1 Standard Output

`-P` <*total space*>
P. The total size of the file system in 512 B units. The exact meaning of this figure is *implementation-defined*, but should include <*space used*>, <*space free*>, and any space reserved by the system not normally available to a user.

S. The value of <*total space*> comes from the *f_blocks* member of the struct `statvfs(2)` function. The `df` utility uses this *f_blocks* member to represent the total number of data blocks in the file system.

`-P` < *space free*>
P. When this figure is less than or equal to zero, it shall not be possible to create any new files on the file system without first deleting others, unless the process has appropriate privileges.

S. "Appropriate privileges" means processes with the effective id of '0' (e.g. root authority).

5.10.7.2.5 `chdir`

P. If **HOME** is empty or is undefined, the default behavior is *implementation-defined.*

S. The `ex` utility prints the following error: No such file or directory.

5.10.7.2.13 `list`

P. Write the addressed lines in a way that should be unambiguous: nonprintable characters shall be written as *implementation-defined* multicharacter sequences...

S. The `ex` utility displays nonprintable characters in the current locale as a circumflex (^) followed by a single character or two uppercase hexadecimal digits. If the nonprintable character is a control character defined in Table 2-20 of the POSIX.2 standard, the corresponding alphabetic character in the table is displayed after the circumflex. Otherwise, the two uppercase hexadecimal digits displayed are those that represent the actual value of the nonprintable character.

5.10.7.2.14 `map`

P. Implementations may restrict the set of commands accepted within *rhs*; the list of restrictions is *implementation-defined*.

S. There are no restrictions on *rhs*.

5.10.7.2.21 `print`

P. Nonprintable characters, except for <`tab`>, shall be written as *implementation-defined* multicharacter sequences.

S. Nonprintable characters are written in octal.

5.10.7.2.29 `source`

P. The maximum supported nesting depth is *implementation-defined*, but shall be at least one.

S. The `ex` utility nesting is limited by the number of `dup(2)` system calls that a single process can have. `dup(2)` is limited via `getrlimit(2)`.

5.10.7.2.37 `write`

P. If *file* is specified and is not the current file, and the file named by *file* exists, then the write shall fail. If the current file has been changed by the `file` command, and that files exist, the write shall fail. In either case, the write can be forced by appending the character `!` to the command name. An existing file can be appended to by appending `>>` to the command name. If the file does not exist, the result is *implementation-defined*.

S. The write succeeds if one has write permission in the directory.

5.10.7.5.8 `list`

P. If `list` is set, write the addressed lines in a way that should be unambiguous: non printable characters shall be written as *implementation-defined* multicharacter sequences; the end of the line shall be marked with a `$`.

S. Nonprintable characters are written in octal.

5.10.7.5.12 `paragraphs, para`

P. The `sections` option can be set to a character string consisting of zero or more character pairs. The default value is *implementation-defined*.

S. The default values are JP, LP, PP, OP, P , LI, pp, lp, ip, np, and bp.

5.10.7.5.18 `sections`

P. The default value is *implementation-defined*.

S. The default values are NH, SH, H , HU, uh, sh, and +c.

5.10.7.5.24 `tags`

P. By default, filenames of `tags` shall be searched for in the current directory and in other *implementation-defined* directories.

S. Filenames of `tags` are also searched for in `/usr/lib/tags`.

5.10.7.5.28 `window`

P. The baud rate of the terminal line may reduce the default in an *implementation-defined* manner.

S. For baud rate less than 1200, terminal lines equal 8. For baud rates greater than or equal to 1200 but less than 2400, terminal lines equal 16.

5.12.2 `fc` – *Process Command History List: Description*

P. When the number reaches an *implementation-defined* upper limit, which shall be no smaller than the value in **HISTSIZE** or 32767 (whichever is greater), the shell may wrap the numbers, starting the next command with a lower number (usually 1).

S. The upper limit is the maximum positive integer or 2147483647.

5.12.5.3 Environment Variables

HISTFILE
P. An implementation may choose to access this variable only when initializing the history file; this initialization shall occur when `fc` or `sh` first attempts to retrieve entries from, or add entries to, the file as the result of commands issued by the user, the file named by the **ENV** variable, or implementation-defined startup files. Therefore, it is *implementation-defined* whether changes made to **HISTFILE** after the history file has been initialized are effective.

S. The **HISTFILE** environment variable is examined when the history file is opened for the first time. If the **HISTFILE** environment variable is changed, then the current history file is closed and a new history file is opened according to the new value of the **HISTFILE** environment variable.

P. Implementations may choose to disable the history list mechanism for users with appropriate privileges who do not set **HISTFILE**; the specific circumstances under which this will occur are *implementation-defined.*

S. There are no circumstances under which the history list mechanisms are disabled.

P. An implementation may choose to access this variable only when initializing the history file, as described under **HISTFILE**. Therefore, it is *implementation-defined* whether changes made to **HISTFILE** after the history file has been initialized are effective.

S. The **HISTFILE** environment variable is examined when the history file is opened for the first time. If the **HISTFILE** environment variable is changed, then the current history file is closed and a new history file is opened according to the new value of the **HISTFILE** environment variable.

5.14.2 `file` – Determine File Type: Description

P. If the file is not a regular file, its file type shall be identified. The file types directory, FIFO, block special, and character special shall be identified as such. Other *implementation-defined* file types may also be identified.

S. The `file` utility uses the information contained in the file named `/etc/magic` to identify the file types stated in the POSIX.2 standard.

5.16.2 `man` – Display System Documentation: Description

P. If more information is available, the `man` utility shall provide it in an *implementation-defined* manner.

S. All the information that `man` provides is displayed in the same manner; as a sequence of text characters obtained from a file found in the directories specified by the user's MANPATH environment variable.

5.16.6.1 Standard Output

P. The `man` utility writes text describing the syntax of the utility *name*, its options, and it operands or, when `-k` is specified, lines from the summary database. The format of this text is *implementation-defined*.

S. The `man` utility displays the contents of a file (as described in section 5.16.2 above) after being processed by the command "`nroff -mansun`".

5.17.2 `mesg` – Permit or Deny Messages: Description

P. Processes with appropriate privileges may be able to send messages to the terminal independent of the current state.

S. *The POSIX 1003.2 Shell and Utility Application Interface* does not allow messages to be sent to a terminal if its state does not permit it.

5.18.3 Options

`-u`
P. Treat <`backspace`> as a printable control character, displayed as an *implementation-defined* character sequence suppressing backspacing and the special handling that produces underlined or standout-mode text on some terminal types.

S. The `more` utility displays the `<backspace>` as the two character string `^H`.

5.18.7 Extended Description

P. It is *implementation-defined* how other nonprintable characters are written.

S. Other nonprintable characters are written as Ctrl-*letter* and ESC-*letter*.

P. In the case that text is being taken from a nonrewinding stream, such as a pipe, it is *implementation-defined* how much backwards motion is supported.

S. The `more` utility sets no limit to the amount of backward motion supported, other than the amount of free space in the `/tmp` directory (or the **TMPDIR** directory, if defined.)

5.18.7.1 Help

P. Write a summary of these command and other *implementation-defined* commands.

S. The `more` utility implements an additional command called `!` which is included in this summary.

5.18.7.24 Invoke Editor

P. It is *implementation-defined* whether line-setting options are passed to editors other than `vi` and `ex`.

S. The `more` utility does not pass line-setting option to any other editors.

P. The file types that can be edited are *implementation-defined.*

S. Only files which are acceptable to the editor being invoked may be edited successfully.

5.19.2 `newgrp` – Change to a New Group: Description

P. If no password is required for the specified group, it is *implementation-defined* whether users not listed as members of that group can change to that group. Whether or not a password is required, *implementation-defined* system accounting or security mechanisms may impose additional authorization restrictions.

S. Access to the group is denied if a group has no password assigned to it and the user is not a member of the group.

5.20.2 `nice` – *Invoke a Utility with an Altered System Scheduling Priority: Description*

P. With no options and only if the user has appropriate privileges, the executed utility shall be run with a system scheduling priority that is some *implementation-defined* quantity less than or equal to the system scheduling priority of the current process.

S. With no options, the `nice` utility increments the system scheduling priority by a value of 10.

P. If the user lacks appropriate privileges to affect the system scheduling priority in the requested manner, the `nice` utility shall not affect the system scheduling priority; in this case, a warning message may be written to the standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

S. The user's ability to alter the system scheduling priority depends on whether or not the process is a "time-sharing" (TS) process and depends on the appropriate privileges required by the `priocntlset(2)` and `priocntl(2)` functions. The `nice` utility only operates on processes that are in the "time-sharing" class and uses the `priocntl(2)` function to get the process current nice value, and then uses `priocntlset()` to increment/decrement this nice value.

5.20.3 Options

`-n` *increment*
P. If the *increment* option argument shall be a positive or negative decimal integer that shall be used to modify the system scheduling priority of the executed utility in an implementation-defined manner.

S. SunOS has a notion of "nice"-ness and this value shall increment or decrement the processes "nice" value. This "nice" value is used to determine the system scheduling priority. This is done using the `priocntlset(2)` function.

5.21.5.2 Input Files

P. The `nm` utility may accept additional *implementation-defined* object library formats for the input file.

S. The `nm` utility does not accept any additional object library formats.

5.21.6.1 Standard Output

P. Symbol type, which shall either be one of the following single characters or an *implementation-defined* type represented by a single character:

`A`	Global absolute symbol
`a`	Local absolute symbol
`B`	Global bss symbol
`b`	Local bss symbol
`D`	Global data symbol
`d`	Local data symbol
`T`	Global text symbol
`t`	Local text symbol
`U`	Undefined symbol

S. The `nm` utility uses the following types:

`B`	Global bss symbol
`b`	Local bss symbol
`D`	Global data symbol
`d`	Local data symbol
`T`	Global text symbol
`t`	Local text symbol
`U`	Undefined symbol
`n`	Locally no defined type
`N`	Globally no defined type

5.23.2 `ps` – Report Process Status: Description

P. When the `-o` option is not specified, information about processes selected shall be written in an *implementation-defined* manner.

S. When the `-f` option is used, process information shall be displayed as if the following was specified on the command line:

-o pid,ppid, c
-o stime,tty=TTY
-o time=TIME
-o args=CMD

When the `-j` option is used, process information shall be displayed as if the following was specified on the command line:

-o pid,ppid,sid
-o time=TIME
-o fname=CMD

When the `-l` option is used, process information shall be displayed as if the following was specified on the command line:

-o pid,tty=TTY
-o time=TIME
-o fname=CMD

5.23.3 Options

`-t` *termlist*
P. Terminal identifiers shall be given in an *implementation-defined* format.

S. Terminal identifiers shall be accepted and displayed in the same format as is stored in the system `/etc/utmp` file which is the same format as is displayed by the `who` utility.

5.23.61 Standard Output

P. When the `-o` option is not specified, the standard output is *implementation-defined.*

S. Without options, `ps` prints information about processes associated with the controlling terminal. The output contains only the process ID, terminal identifier, cumulative execution time, and the command name.

`args`
P. The implementation may truncate this value to the field width; it is *implementation-defined* whether any further truncation occurs.

S. The maximum number of characters displayed is limited to 80.

P. Any *implementation-defined* variables shall specify in the conformance document if the field may contain <`blank`>s as well as the default header.

S. No implementation-defined variables contain <`blank`>s.

Table 8-1 lists the Solaris implementation format specifiers and the default header used with each.

Table 9-1 Solaris Implementation Format Specifiers and their Default Header

Format Specifier	Default Header	Format Specifier	Default Header
`addr`	`ADDR`	`pri`	`PRI`
`c`	`C`	`rgid`	`RGID`
`class`	`CLS`	`rss`	`RSS`
`f`	`F`	`ruid`	`RUID`
`fname`	`COMMAND`	`s`	`S`
`gid`	`GID`	`sid`	`SID`
`opri`	`PRI`	`stime`	`STIME`
`osz`	`SZ`	`uid`	`UID`
`pmem`	`%MEM`	`whan`	`WCHAN`

5.24.2 `renice` *– Set System Scheduling Priorities of Running Processes: Description*

P. The system scheduling priority shall be bounded in an *implementation-defined* manner. If the requested increment...would raise or lower the system scheduling priority of the executed utility beyond *implementation-defined* limits, then the limit whose value was exceeded shall be used.

S. See section 5.20.3 *Options* {*of* `nice`} in this document.

P. Regardless of which options are supplied or any other factor, `renice` shall not alter the system scheduling priorities of any process unless the user requesting such a change has appropriate privileges to do so for the specified process. If the user lacks appropriate privileges to perform the requested action, the utility shall return an error status.

S. See section 5.20.2 *Description* {*of* `nice`} in this document.

5.24.3 Options

`-n` *increment*
P. Negative *increment* values may require appropriate privileges and shall cause a higher system scheduling priority.

S. See section 5.20.3 *Options* {*of* `nice`} in this document.

5.26.2 `strings` – Find Printable Strings in Files: Description

P. Additional *implementation-defined* strings may be written.

S. No other strings are written.

5.26.3 Options

`-a`
P. If `-a` is not specified, it is *implementation-defined* what portion of each file is scanned for strings.

S. The `strings` utility ignores the portion of an executable file containing executable instructions.

5.28.2 `talk` – Talk to Another User: Description

P. When and only when the `stty` *iexten* local mode is enabled, additional special control characters and multibyte or single-byte functions shall be processed in an *implementation-defined* manner.

S. The `talk` utility does not support additional special control characters and multibyte or single-byte functions.

P. Typing other nonprintable characters shall cause *implementation-defined* sequences of printable characters to be sent to the terminal of the recipient.

S. Other nonprintable characters are displayed on recipient's terminal as "?" character.

P. However a user's privilege may further constrain the domain of accessibility of other user's terminals. The `talk` utility shall fail when the user lacks the appropriate privileges to perform the requested action.

S. The `talk` utility does not further constrain the domain of accessibility of other users terminals beyond that imposed by the `mesg` utility.

5.30.4 Operands

`init`
P. Display the sequence that will initialize the terminal of the user in an *implementation-defined* manner.

S. The `tput` utility, the TERMINFO database defines this sequence for each terminal.

`reset`
P. Display the sequence that will reset the terminal of the user in an *implementation-defined* manner.

S. The `tput` utility, the TERMINFO database defines this sequence for each terminal.

5.36.2 `who` – Display Who is on the System: Description

P. The `who` utility shall list various pieces of info about accessible users. The domain of accessibility is *implementation-defined.*

S. The `who` utility shows information for all users.

5.36.6.1 Standard Output

P. The `who` utility writes its default information to the standard output in an *implementation-defined* format, subject only to the requirement of containing the information [in the standard].

S. The `who` utility writes its default information to the standard output in the following format:

`%s %s %s %s`, <*user name*>,<*terminal name*>,<*time of login*>

5.37.2 `write` – Write to Another User: Description

P. When and only when the `stty` *iexten* local mode is enabled, additional special control characters and multibyte or single-byte functions shall be processed in an *implementation-defined* manner.

S. The `write` utility does not support any additional special control characters or multiple or single-byte functions.

P. Typing other nonprintable characters shall cause *implementation-defined* sequences of printable characters to be written to the terminal of the recipient.

S. Other nonprintable characters are displayed on recipient's terminal as "?" character.

P. To write to a user who is logged in more than once, the terminal argument can be used to indicate which terminal to write to; otherwise, the recipient's terminal is selected in an *implementation-defined* manner and an informational message shall be written to the sender's standard output, indicating which terminal was chosen.

S. The terminal selected is taken from the first entry in the `/etc/utmp` file that contains the user id that matches the recipients user id.

P. However, a privilege of a user may further constrain the domain of accessibility of the terminals of other users. The `write` utility shall fail when the user lacks the appropriate privileges to perform the requested action.

S. The `write` utility does not further constrain the domain of accessibility of other users terminals beyond that imposed by the `mesg` utility.

POSIX.2 Section 6, Software Development Utilities Option

6.2.7.1 Makefile Syntax

P. If `./makefile` is not found, the file `./Makefile` shall be tried. If neither `./makefile` nor `./Makefile` are found, other *implementation-defined* pathnames may also be tried.

S. The following pathnames are tried in the order given:

./makefile
./Makefile
s.makefile
s.Makefile
SCCS/s.makefile
SCCS/s.Makefile

6.2.7.2 Makefile Execution

P. The macros from the command line to `make` shall be added to the environment of `make`. Other *implementation-defined* variables may also be added to the environment of `make`.

S. The **MAKEFLAGS** variable is added if it did not already exist.

P. If the **MAKEFLAGS** variable is not set in the environment in which `make` was invoked, in the makefile, or on the command line, it shall be created by `make` and shall contain all options specified on the command line except for the `-f` and `-p` options. It may also contain *implementation-defined* options.

S. The make utility, **MADEFLAGS** may also contain the `-E`, `-V`, `-v`, and `-x` options, if they appeared on the original make command line. See the `man` page for `make`, in the SUNWp2man package, for a description of these options.

6.2.7.3 Target Rules

P. The interpretation of targets containing the characters "%" and " `"` " is *implementation-defined.*

S. The `make` utility treats targets containing "%" as meta-rules unless the user specifies the `.POSIX` special target, in which case, it ignores meta-rules. The make utility uses the " `"` " character in pairs for quoting. It treats a # character contained within a " `"` " pair as though it has no special meaning.

6.2.7.4 Macros

P. If `SHELL` is defined in the makefile or is specified on the command line, it shall replace the original value of the `SHELL` macro, but shall not affect the **SHELL** environment variable. Other effects of defining `SHELL` in the makefile or on the command line are *implementation-defined.*

S. If the user specifies the `.POSIX` special target, the effect of defining `SHELL` is as specified in the POSIX.2 standard. If the user does not specify `.POSIX` and defines `SHELL` as a macro in the makefile, `make` uses the shell specified by `SHELL` but does not change the value of the **SHELL** environment variable in the environment passed to child processes unless the user specified the `-x` option. If the user does not specify `.POSIX` and includes `SHELL=`*shell_path* on the command line, `make` uses *shell_path* as its shell and also assigns it as the value of the **SHELL** environment variable in the environment passed to child processes.

POSIX.2 Annex A: C Language Development Utilities Option

A.1.3 Options

`-D`
P. Additional *implementation-defined names* may be provided by the compiler.

S. The following additional names are provided by the C compiler:

unix
sparc (SPARC only)
i386 (x86 only)
sun

The above are not pre-defined in -Xc mode.

The following predefinitions are valid in all modes:

_sparc (SPARC only)
_i386 (x86 only)
_unix
_sun
_BUILTIN_VA_ARC_INCR
_SUNPRO_C=0x301
_SVR4

A.1.4 Operands

file.`a`
P. Implementations may recognize *implementation-defined* suffixes other than `.a` as denoting object file libraries.

S. None

file`.o`
P. Implementations may recognize *implementation-defined* suffixes other than `.o` as denoting object file libraries.

S. None

`-L` *library*
P. Implementation may recognize *implementation-defined* suffixes other than `.a` as denoting libraries.

S. None

A.1.5.2 Input Files

P. Additional input file formats are *implementation-defined.*

S. The C compiler recognizes the additional input file format: `.il` files (inline template files.)

A.1.7.2 External Symbols

P. The C compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; the action taken upon encountering symbols exceeding the *implementation-defined* maximum symbol length is unspecified.

S. The C compiler supports up to 1023 characters.

P. The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols total. A diagnostic message shall be written to the standard output if the *implementation-defined* limit is exceeded; other actions are unspecified.

S. The number of symbols allowed by the C compiler per source file is dynamic and only limited to the memory and disk space available on the system.

A.2.6.1 Standard Output

P. If the `-t` option is not specified: (1) *implementation-defined* informational, error, and warning messages concerning the contents of `lex` source code input shall be written to either the standard output or standard error.

S. The `lex` utility writes all information, error and warning messages to the standard error. For details of the text of those messages, see section A.2.6.2 Standard Error.

P. If the `-v` option is specified and the `-n` option is not specified, `lex` statistics also shall be written to either the standard output or standard error, in an *implementation-defined* format.

S. The `lex` utility writes these statistics to the standard error. For details of the format of those statistics, see section A.2.6.2 Standard Error.

A.2.6.2 Standard Error

P. If the `-t` option is specified, *implementation-defined* informational, error, and warning messages concerning the contents of `lex` source code input shall be written to standard error.

S. The lex utility writes all information, error and warning messages to the standard error. The following list shows these messages:

```
"Error: EOF in string or character constant"
"Error: EOF inside comment"
"Error: Non-terminated string or character constant"
"Error: Unexpected EOF inside comment"
"Error: Action does not terminate"
"Error: Can't open %s"
"Error: Cannot open file -- %s"
"Error: Cannot read from -- %s"
"Error: Character %o used twice"
"Error: Character range specified between different codesets."
"Error: Character value %d out of range"
"Error: Definitions too long"
"Error: EOF before %%%%"
"Error: EOF inside comment"
"Error: Illegal definition"
"Error: Illegal rule"
"Error: Invalid request %s"
"Error: Non-ASCII characters in start condition."
```

```
“Error: None-ASCII characters in start condition.”
“Error: Parse error”
“Error: Parse tree too big %s”
“Error: Premature EOF”
“Error: Start conditions too long”
“Error: Too complex rules -- requires too many char groups.”
“Error: Too late for %array”
“Error: Too late for %pointer”
“Error: Too late for language specifier”
“Error: Too many (>%d) pattern-action rules.”
“Error: Too many characters pushed”
“Error: Too many definitions”
“Error: Too many large character classes”
“Error: Too many packed character classes”
“Error: Too many positions %s”
“Error: Too many start conditions used”
“Error: Too many start conditions”
“Error: Too many states %s”
“Error: Too many transitions %s”
“Error: \Character table (%t) is supported only in ASCII
    compatibility mode.\n”
“Error: bad translation format
“Error: can‘t have negative iteration”
“Error: ch table needs redeclaration”
“Error: definition %ws not found”
“Error: definition too long”
“Error: definitions too long”
“Error: executable statements should occur right after %%%%”
“Error: illegal extra \”}\””
“Error: illegal extra slash”
“Error: illegal number of packed character class”
“Error: illegal number of parse tree nodes”
“Error: illegal operator -- %c”
“Error: illegal position number”
“Error: illegal size of output array”
“Error: illegal state number”
“Error: illegal translation number”
“Error: incomplete translation format”
“Error: iteration range must be positive”
“Error: missing translation value”
“Error: output table overflow”
“Error: string name too long”
“Error: unacceptable statement”
“Error: undefined action string”
“Error: undefined start condition %ws”
“Error: unexpected EOF before %%%%”
```

```
“Error: unmatched hyphen”
“Warning: Character ‘%wc’ used twice”
“Warning: No translation given - null string assumed”
“Warning: Non-portable Character Class”
“Warning: Non-terminated string”
“Warning: String too long”
“Warning: Substitution strings may not begin with digits”
“Warning: \”%c\” redefined inside brackets”
“Warning: \\a is ANSI C \”alert\” character”
“Warning: \\x is ANSI C hex escape”
“Warning: bad state %d %o”
“Warning: bad transition %d %d”
“Warning: invalid string following %%%% be ignored”
“Warning: string too long”
“Warning: the values between braces are reversed”
“Warning: undefined string”
```

P. If the `-t` option is not specified:

`(1)` *implementation-defined* informational, error, and warning messages concerning the contents of `lex` source code input shall be written to either the standard output or standard error.

S. The `lex` utility writes all information, error and warning messages to the standard error. See section A.2.6.2 for a list of these messages.

P. (2) If the `-v` option is specified and the `-n` option is not specified, `lex` statistics also shall be written to either the standard output or standard error, in an *implementation-defined* format.

S. The `lex` utility writes these statistics to the standard error. The following list shows these errors:

```
“%d/%d nodes(%%e), %d/%d positions(%%p), %d/%d (%%n),
    %ld transitions, \n”
“%d/%d packed char classes(%%k),”
“%d/%d packed transitions(%%a),”
%d/%d output slots(%%o) \n”
```

A.2.7 Extended Description

P. The input string that was matched is left in *yytext* as a null-terminated string; *yytext* is either an external character array or a pointer to a character string. As explained in A.2.7.1, the type can be explicitly selected using the `%array` or `%pointer` declarations, but the default is *implementation-defined*.

S. By default, the `lex` utility behaves as if the user had explicitly selected the %array declaration.

A.2.7.1 `lex` Definitions

P. The default type of `yytest` is *implementation-defined*.

S. The default type of yytext is `char[]`.

P. In [Table A-1], *n* represents a positive decimal integer, preceded by one or more <`blank`>s. The exact meaning of these table size numbers is *implementation-defined*. The implementation shall document how these numbers affect the `lex` utility and how they are related to any output that may be generated by the implementation should space limitations be encountered during the execution of `lex`.

S. The following table size declarations represent settable limits: `%p`, `%n`, `%a`, `%e`, `%k`, and `%o`. The lex statistics can be used to show the table sizes, which are set by default using the values shown in table

Table 9-2

Declaration	Description	Default
%p n	number of positions	2500
%n n	number of states	500
%a n	number of transitions	2000
%e n	number of parse tree nodes	1000
%k n	number of packed character classes	10000
%o n	size of the output array	3000

Depending on the system configuration and the available resources, limits may affect what input `lex` can successfully compile.

A.2.7.4 `lex` Regular Expressions

Table A-2 \digits
P. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is *implementation-defined*.

S. The `lex` utility does not support byte sizes greater than 8 bits.

A.3.6.3.3 `yacc` Description File

P. Limits for internal tables also shall be reported in an *implementation-defined* manner.

S. The `yacc` expands the internal tables as needed. The only limitation is system memory.

A.3.7.9 Limits

P. The exact meaning of these values is *implementation-defined*. The *implementation shall define* the relationship between these values and between them and any error messages that the implementation may generate should it run out of space for any internal structure.

S. The internal tables are allocated dynamically. When system memory gets short and allocation of memory fails, one of the following error messages is issued:

“Couldn’t allocate initial table”
“Could not allocate lookset array”
“Cannot allocate tables in mktbls”

Other Standards 10

This chapter discusses the conformance of Solaris to prevailing standards.

ANSI C Programming Language

The need for a single clearly defined C standard arose as use of the C programming language expanded rapidly and a variety of differing translator implementations were being developed. The American National Standard Programming Language C addressed the problems this need posed to the developer and the implementor by specifying the C language precisely.

The ANSI C standard specifies the syntax and semantics of programs written in the C programming language. It specifies the C program's interaction with the execution environment through input and output data. It also specifies restrictions and limits imposed upon conforming implementations of C language translators.

The standard was developed by the X3J11 Technical Committee on the C Programming Language under project 381-D by the American National Standards Committee on Computers and Information Processing (X3). The work of X3J11 began in the summer of 1983, based on several documents that were made available to the Committee. The Committee divided the effort into three pieces: the environment, the language and the library, and each of these areas is addressed in the standard.

Note – The use of American National Standards is completely voluntary.

Compliance With the ANSI C Standard

Sun ANSI C is fully compliant with the ANSI C standard.

ANSI C Specification and Related Publications

The first manual listed below is the ANSI C standard specification. The second and third manuals listed are part of the Solaris documentation set. The *SPARCompiler C Transition Guide* describes techniques for writing new and upgrading existing C code to comply with the ANSI C language specification.

- *American National Standard for Information Systems Programming Language C*, American National Standards Institute
- *SPARCompiler C 4.0 Transition Guide for SPARC Systems*–Sun Microsystems
- *SPARCompiler C 4.0 User's Guide*–Sun Microsystems

ANSI/IEEE 754

The ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic is a product of the Floating-Point Working Group of the Microprocessor Standards Subcommittee of the IEEE Computer Society. The standard defines a family of commercially feasible ways for systems to perform binary floating-point arithmetic. The issues of retrofitting were not considered when the standard was defined; instead, the interests of the user community were placed above the goal of industrial continuity at that time.

There are three major aspects to the standard: the format of data types, the arithmetic and the exception handling. The objective of the standard is that an implementation of a floating-point system conforming to it could be realized entirely in software, entirely in hardware, or in any combination of hardware and software.

Compliance With ANSI/IEEE 754

Sun FORTRAN 2.0.1 conforms to ANSI/IEEE Std. 754-1985.

ANSI/IEEE 754-1985 Specification and Related Publications

The first document listed below is the IEEE 754 Standard. It is followed by a Sun publication that discusses the standard.

- *IEEE Standard for Binary Floating-Point Arithmetic,* ANSI / IEEE Std. 754-1985.
- *Numerical Computation Guide*, Sun Microsystems, Inc., 1991
- *A Proposed Standard for Binary Floating-Point Arithmetic,* IEEE COMPUTER, March 1981

International Standards Organization (ISO) 8859-1

ISO 8859 consists of several parts, each of which specifies a set of up to 191 graphic characters and the coded representation of each of these characters by means of a single 8-bit byte. Each set is intended for use for a group of languages.

ISO 8859, Part 1 specifies a set of 191 graphic characters identified as Latin alphabet No. 1. The set of graphic characters comprising Latin alphabet No. 1 is intended for use in data processing and text applications and may also be used for information interchange.

A set of graphic characters is considered in conformance with ISO 8859 if it comprises all graphic characters declared in the specification to the exclusion of any other, and if their coded representations are those specified by ISO 8859.

Compliance With ISO 8859-1

Solaris is entirely compliant with the ISO 8859-1 standard.

ISO 8859 Standard

- *International Standard ISO 8859-1*

Federal Information Processing Standard (FIPS) 151

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce.

The FIPS 151 standard is called the Kernel Operations Component of the Applications Portability Profile (APP). FIPS 151 is part of a series of FIPS for the APP.

FIPS 151-2, which corresponds to IEEE Std. 1003.1 - 1990 was ratified by NIST on October 15, 1993. It supersedes FIPS 151-1 (which corresponded to IEEE Std. 1003.1 - 1988) in its entirety as the POSIX.1 reference standard.

Compliance With FIPS 151

Solaris 2.5 conforms to FIPS 151-2 on several SPARC and x86 platforms.

FIPS 151 Specification

- *The Federal Information Processing Standards Publication*, National Institute of Standards and Technology

Federal Information Processing Standard (FIPS) 158

The FIPS 158 standard is called the User Interface Component of the Applications Portability Profile (APP).

The functional components of FIPS 158 constitute a toolbox of standard elements that can be used to develop and maintain portable applications. FIPS 158 is the first step in responding to a need within the federal community for a set of tools to develop standard user interfaces. FIPS 158 is based upon the X Window System developed by the X Consortium. The X Window System assumes a client/server model of distributed computing and user interface applications based upon bit-mapped graphic displays.

The FIPS 158 standard adopts the specifications for X Version 11, Release 3 (X11R3). These specifications consist of the documents for the X Window System Protocol, X Version 11: the Xlib-C language X Interface (Xlib), the X

Toolkit Intrinsics-C Language Interface (Xt) and the Bitmap Distribution Format 2.1. The interfaces specified in FIPS 158 represent the consensus of the industry for lower-level X Window System interfaces.

Compliance With FIPS 158

OpenWindows, the Solaris windowing environment, conforms to FIPS 158 by fully implementing X11 (Xlib) and the X11 protocol.

The OpenWindows OPEN LOOK Intrinsics Toolkit (OLIT) API is an implementation of MIT's Xt toolkit (Xt intrinsics, Version R5) with an OPEN LOOK widget set. OLIT is composed of prebuilt components that fit into intrinsics applications. OLIT conforms with the Xt intrinsics toolkit; because X11, Release 5 is backwardly compatible with X11, Release 4, OLIT conforms to X11, Release 4.

OpenWindows fully supports ICCCM, which provides basic policy on rules for transferring data between applications, transfer of keyboard focus, layout schemes and colormap installation.

FIPS 158 Specification and Related Publications

- *The Federal Information Processing Standards Publication; The User Interface Component of the Applications Portability Profile*, issued by the National Institute of Standards and Technology, October, 1992.
- *Solaris OpenWindows User's Guide*, SunSoft Press

The Application Binary Interface (ABI)

The Application Binary Interface (ABI) defines the binary system interface between compiled applications and the operating system on which they run. The ABI provides binary portability across UNIX System V Release 4 platforms sharing the same CPU architecture.

The System V Application Binary Interface continues to evolve to address new technology and market requirements and is reissued at intervals of approximately three years. Each new edition of the specification is likely to contain extensions and additions that will increase the potential capabilities of applications that are written to conform with the ABI.

Compliance With the ABI

It is the intention of SunSoft to comply with the ABI as it evolves.

ABI Publication

- *AT&T System V Application Binary Interface: Generic ABI and Application Binary Interface SPARC Processor Supplement* - Prentice-Hall.
- *AT&T System V Application Binary Interface Intel 386 Processor Supplement*

SPARC Compliance Definition (SCD)

The SPARC Compliance Definition (SCD) is a formal specification of the system hardware and software to be met by manufacturers of SPARC systems to ensure that those systems run compliant applications. The SCD also details specific interfaces that can be safely used by an application with assurance that the application binary will run on all compliant SPARC hardware platforms.

The SCD specification was developed by members of SPARC International (SI). SI is now responsible for administering usage of the SPARC trademark to compliant systems.

Sun Microsystems and SunSoft worked with SI to develop SCD 2.1 which is closely connected to SVR4 and the SPARC ABI specification.

Compliance With the SCD

Systems produced by Sun Microsystems and SunSoft are fully compliant with SCD 2.1.

SPARC Compliance Definition Specification

- *SPARC Compliance Definition 2.1* - SPARC International

Index

Adobe PostScript