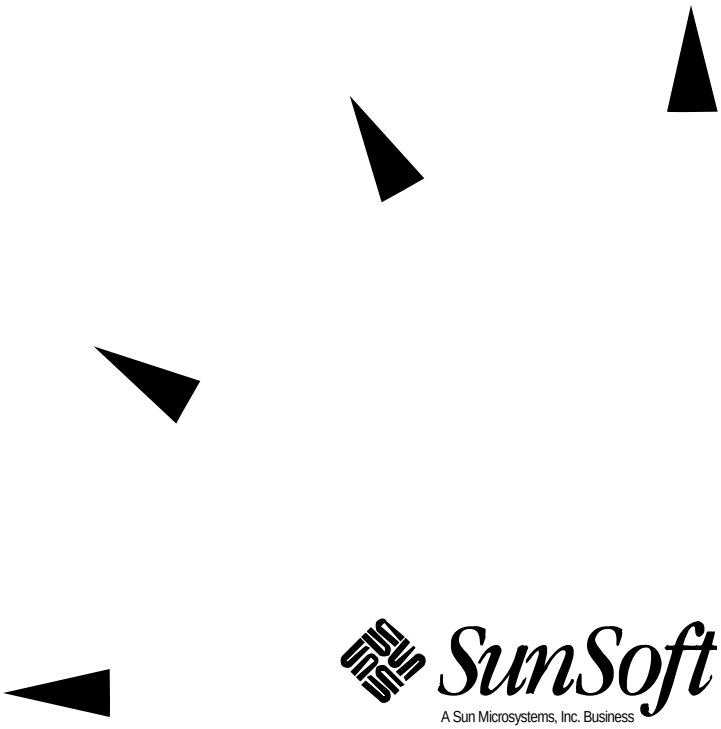
Standards Conformance Reference Manual



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

SunSoft
A Sun Microsystems, Inc. Business

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 2.4 and the SunOS 5.4 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)
- X Window System™ Protocol, Version 11 (X11)
- X / Open™ XPG3 BASE
- X / Open XPG4 BASE
- ANSI / IEEE Standard 1003.1–1990 – (POSIX.1)
- ANSI C Programming Language
- International Standards Organization (ISO) 8859-1
- Federal Information Processing Standard 151 (FIPS 151)
- Federal Information Processing Standard 158 (FIPS 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 7 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 Standards 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).

Chapter 6 of this guide discusses the compliance of Solaris software with IEEE Standard 1003.1-1990.

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 Version 3 (XPG3) encompasses the IEEE POSIX.1 operating system interface and numerous extensions. Version 4 includes POSIX.2.

Chapter 4 of this guide discusses the compliance of Solaris to the programming interface specifications presented in the X/Open Portability Guide, Issue 3.

Chapter 5 of this guide discusses the compliance of Solaris to the programming interface specifications presented in the X/Open Portability Guide, Issue 4.

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.

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.

1

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.

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

This chapter provides an introduction to UNIX System V Release 4 (SVR4), discusses related specifications and identifies how the SunOS operating system and OpenWindows conform to those specifications.

UNIX System V Release 4 (SVR4)

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.

2

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.

2

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.

The SVID is compliant with IEEE Std. 1003.1-1990 (POSIX.1) and will continue to evolve towards compliance with other appropriate industry standards as they are approved.

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.*

2

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.)

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 has become the standard look and feel for bit-mapped displays under UNIX System V Release 4.

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 2.4. 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 2.4, OLIT is Level 2 compliant with certain exceptions.
- XView 3.3 Toolkit (X11-based Visual/Integrated Environment for Workstations): XView is an X11 toolkit for building applications. The XView API is is based upon Xlib, the lowest level of programming available to the X window system programmer. XView implements the OPEN LOOK GUI. For Solaris 2.4, 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.

X11, PostScript and Sun Microsystems' OpenWindows 3

This chapter discusses the X Window System, Version 11 (X11) and the parts of OpenWindows 3.4 that implement aspects of X11.

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 Reference Manual*, Sun Microsystems, Inc.
- *Desktop Integration Guide*, Sun Microsystems, Inc.

X/Open and XPG3 4

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 2.4 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.

4

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 5,“X/Open and XPG4”.

This chapter identifies Solaris 2.4 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 4-1 The XPG3 Base Brand Logo

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.

4

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

Chapter 2: INTERNATIONALIZED SYSTEM CALLS AND HEADERS

PRODUCT IDENTIFICATION

Product Identification	Solaris
Version/Release No.	2.4

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	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 behaviour and any test results to be reproduced.

SPARC

SPARC running Solaris 2.4. 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: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 2.4. Installation procedures are provided in x86: Installing Solaris Software.

To reproduce the test environment, do 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 of 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 file 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

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
`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

(continued)

Function	Error	Detected
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
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

(continued)

Function	Error	Detected
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.

2.1.5 MATHEMATICAL INTERFACES

Question 6: *What format of floating point numbers are 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. The decoding routine is not provided outside the United States.

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:

See POSIX.1 Conformance Statement in Chapter Five of the *Standards Conformance Guide*.

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 the 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 on 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 behaviour 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 behaviour specified for an implementation that supports **job control**. This later 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*

See POSIX.1 Conformance Statement in Chapter Five of the *Standards Conformance Guide.*

Chapter 3: COMMANDS AND UTILITIES

PRODUCT IDENTIFICATION

Product Identification	Solaris
Version / Release No.	2.4

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 behaviour 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 EXPECTATIONS

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 BEHAVIOUR

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 BEHAVIOUR

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 are 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	Behaviour Specified in XPG3	Supported
ar	LC_TIME affects date format	Yes
awk	LC_COLLATE, LC_CTYPE affect regular expression matching	No
	LC_COLLATE affects the behaviour of string comparisons	No
	LC_NUMERIC affects the behaviour 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 affect regular expression matching	No
	LC_CTYPE is used to determine whether characters are printable	Yes
egrep	LC_COLLATE, LC_CTYPE affect regular expression matching	No
	LC_CTYPE is used to determine character classification (alphabetic, upper case, lower case)	Yes
expr	LC_COLLATE, LC_CTYPE affect regular expression matching	No

(continued)

Command	Behaviour Specified in XPG3	Supported
	LC_COLLATE affects the behaviour 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 affect filename pattern matching	No
grep	LC_COLLATE, LC_CTYPE affect 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 affect filename pattern matching	No
	LC_TIME affects date format	Yes
pg	LC_COLLATE, LC_CTYPE affect 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

(continued)

Command	Behaviour Specified in XPG3	Supported
rm,rmdir	LANG affects yes string	No
sed	LC_COLLATE, LC_CTYPE affect regular expression matching	No
	LC_CTYPE is used to determine whether a character is printable	Yes
sh	LC_COLLATE, LC_CTYPE affect filename pattern matching	No
	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 affect 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 behaviour 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	SunSoft, A Sun Microsystems, Inc. Business
Address	2550 Garcia Avenue Mountain View CA 94043

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 3.205
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 behaviour 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 of 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.

No temporary waivers needed.

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.	2.4

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 behaviour 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 of 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 behaviour 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 of 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.	2.4

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 behaviour 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 of 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

15.1 UTILITIES

PRODUCT IDENTIFICATION

Product Identification	Solaris
Version / Release No.	2.4

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 behaviour 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 2.4. 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 in respect of 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 discs 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
1600 bpi PE magnetic tape	Yes

Rationale

XPG3 states that standards are referenced for transfer of floppy discs 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 that 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`

Options

A definition of the commands used to create and read these formats. If a special option is required to produce the specified format this must be detailed.

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 FILE 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*

4

X/Open and XPG4 5

This chapter discusses the compliance of Solaris 2.4 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 2.4 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.

5

Figure 5-1 The XPG4 Base Brand Logo

Because it is a comforming implementation of XPG3 and meets the requirements of the System Interfaces and Headers component of XPG4, Solaris has earned the XPG4 BASE brand. 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.
- **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.
- **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

Section 1: INTERNATIONALIZED SYSTEM CALLS AND LIBRARIES

PRODUCT IDENTIFICATION

Product Identification	Solaris
Version/Release No.	2.4

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

5

INDICATOR OF COMPLIANCE

VSX Test Suite Release	VSX 4.3.2
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 behaviour and any test results to be reproduced.

SPARC

SPARC running Solaris 2.4. 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: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_scpg4.0`

---

x86

x86 running Solaris 2.4. Installation procedures are provided in x86: Installing Solaris Software.

To reproduce the test environment, do 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_scpg4.0`

TEMPORARY WAIVERS

List below references to any temporary waivers granted by X/Open in respect of 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 1.1: GENERAL ATTRIBUTES

1.1.1 XPG4 FEATURE GROUPS

Question 1: *Which of the following feature groups are supported by the implementation?*

Answer: Solaris 2.4 supports the Shared Memory, Encryption and Enhanced Internationalization feature groups.

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?*

Answer:

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

Rationale

For an X/Open conforming 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 Specfication, 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?*

Answer:

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
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 long 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 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 normalised float.	-125
DBL_MIN_EXP	Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized double.	-1024

(continued)

Macro Name	Meaning	Value
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
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

(continued)

Macro Name	Meaning	Value
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
LDBL_EPSILON	Difference between 1.0 and the least value greater than 1.0 that is representable as a long double.	1.92592994438723585305597794258492731 9E-34L
FLT_MIN	Minimum normalized postive float.	1.175494351E-38F
DBL_MIN	Minimum normalized positive double.	2.2250738585072014E-308
LDBL_MIN	Minimum normalized positive long double.	3.362103143112093506262677817321752260 3E-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)?*

Answer:

Macro Name	Meaning	Minimum	Maximum
ARG_MAX	Max. length of argument to the exec functions 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
NAME_MAX	Max. number of bytes allowed in a terminal input queue.	14	See note
OPEN_MAX	Max. number of files open in a process.	16	See note
PATH_MAX	Max. number bytes in a pathname.	255	See note
PIPE_BUF	Max. number of bytes guaranteed to be atomic in a write to a pipe.	512	5120
STREAM_MAX	Number of streams one process can have open at one time.	8	8
TZ_NAME_MAX	Max. number number of bytes supported for a time zone name.	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 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

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?*

Answer:

Macro Name	Meaning	Min.	Maximum
BC_BASE_MAX	Max. *ibase* and *obase* values allowed by the *bc* utility.	99	32767 (SHRT_MAX)
BC_DIM_MAX	Max. number of elements permitted in an array by the *bc* utility.	2048	32768 (SHRT_MAX+1)
BC_SCALE_MAX	Max. scale value allowed by the *bc* utility.	99	32767 (SHRT_MAX)
BC_STRING_MAX	Max. length of a string constant accepted by the *bc* utility.	1000	2048 (LINE_MAX)
COLL_WEIGHTS_MAX	Max. 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	Max. number of expressions that can be nested within parentheses by the *expr* utility.	32	32
LINE_MAX	Max. 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	Max. number of simultaneous supplementary group IDs per process.	16	16
RE_DUP_MAX	Max. number of repeated occurrence 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 X/Open 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 miminum 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?*

Answer:

Macro Name	Meaning	Value
CHAR_BIT	Number of bits in a char.	8
CHAR_MAX	Max. value of a char.	127
INT_MAX	Max. value of an int.	2147483647
LONG_BIT	Number of bits in a long int.	32

(continued)

Macro Name	Meaning	Value
LONG_MAX	Max. value of a long int.	2147483647L
MB_LEN_MAX	Max. number of bytes in a character, for any supported locale.	5
SCHAR_MAX	Max. value of a signed char.	127
SHRT_MAX	Max. value of a short.	32767
SSIZE_MAX	Max. value of an object of type ssize_t.	2147483647
UCHAR_MAX	Max. value of an unsigned char.	255
UINT_MAX	Maximum value of an unsigned int.	4294967295U
ULONG_MAX	Max. value of an unsigned long int.	4294967295UL
USHRT_MAX	Max. 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?*

Answer:

Macro Name	Meaning	Value
FOPEN_MAX	Number of streams which the implementation guarantees can be open simultaneously.	20 (SPARC) 60 (x86)
L_tmpnam	Max. 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 provide 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?*

Answer:

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

(continued)

Function	Error	Detected
atan()	EDOM	Yes
	ERANGE	No
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

(continued)

Function	Error	Detected
exec()	ENOMEM	Yes
	ETXTBSY	No
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

(continued)

Function	Error	Detected
fputc()	ENOMEM	Yes
	ENXIO	Yes
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

(continued)

Function	Error	Detected
getpwnam()	EIO	Yes
	EINTR	No
	EMFILE	Yes
	ENFILE	Yes
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

(continued)

Function	Error	Detected
log()	EDOM	Yes
	ERANGE	No
log10()	EDOM	Yes
	ERANGE	No
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

(continued)

Function	Error	Detected
sigismember()	EINVAL	Yes
signal()	EINVAL	Yes
sin()	EDOM	No
	ERANGE	Yes
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

(continued)

Function	Error	Detected
write()	ENXIO	Yes
y0()	EDOM	Yes
	ERANGE	No
y1()	EDOM	Yes
	ERANGE	No
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 Mathmatical Interfaces

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

Answer:

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?*

Answer:

Function	Provided
`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 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 files types (regular, directory, FIFO, special, and so on) are considered to be executable?*

Answer:
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?*

Answer: 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?*

Answer:
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?*

Answer:

Yes.

Rationale

This behaviour is ofter provided on systems with mathematical functions that produce these results.

Reference

X/Open CAE Specification, System Interfaces Definitions, 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?*

Answer:

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?*

Answer:

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.

POSIX.1 6

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 Standard 1003.1b

IEEE Std 1003.1b–1993 is a standard that amends POSIX.1 to include extensions in support of realtime applications. (In earlier versions of this guide, IEEE Std 1003.1b was referred to as IEEE Std 1003.4 or POSIX.4. IEEE Std 1003.1b is now the preferred reference to this standard.) These extensions were ratified by the IEEE in September 1993. The functionality of SunOS 5.4 is intended to provide compliance and support of POSIX.1 as amended by IEEE Std 1003.1b–1993.

6

Scope

To comply with Section 1.3.1.2 (*Documentation*), this chapter describes the behavior of features in the SunOS 5.4 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 5.4 operating system framework.

C Standard Compliance

The SunOS 5.4 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 5.4 operating system implements that area.

- **P.** stands for POSIX.
- **S.** stands for SunOS 5.4.

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. To configure Solaris to run with the behavior specified by the standard, execute the following steps:

1. Edit the `/etc/saf/zsmon/_pmtab` file to turn off the `ttysoftcarrier` detect by changing the `ttya` and `ttyb` fields from `:y:` to `:n:`. (The colons (:) act as field separators).
2. In the last line of the `/kernel/drv/options.conf` file, revise the `:bd:` entry to `:cbd:` (The colons (:) act as field separators).
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

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 5.4 operating system is a conforming implementation as defined in Section 1.3.1.2 (*Documentation)* of the IEEE Std 1003.1-1990 *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. Appropriate privileges depend on file permissions and the user ID that executed the process; 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.

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 implementaton.
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.
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.
RTSIG_MAX	Undefined. Value may be configurable in future.	Maximum number of realtime 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.

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	Realtime 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_VERSION	199309L	Solaris conforms to IEEE Standard 1003.1-1990, as amended by the IEEE Standard for Realtime Extensions approved Sept. 1993.

_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 5.4 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. The number of bytes available for the new process's combined argument and environment lists is {ARG_MAX}. The *implementation shall specify* in the system documentation whether any combination of null terminators, pointers, or alignment bytes are included in this total.

S. {ARG_MAX} is retrieved using the `sysconf()` function. This value includes the total number of bytes available for a new process' arguments, environment, and stack. {ARG_MAX} also includes initial pointers into the argument and environment vectors.

The space required for the arguments, environment and stack by an `execve()` call is determined by the following formula:

`space` = (((*na* + 4) * *bpw*) + *nc*) + *click*

space is then rounded up to the next click boundary. A click is the number of bytes that the system's memory-management facilities treats as a single unit. For all machines on which the SunOS 5.4 operating system is supported, one click equals 4096 bytes.

na is the count of arguments and environment variables. *bpw* is the number of bytes per word: 4 on all Sun systems. *nc* is the count of bytes in the argument and the environment vectors, including null terminators, and rounded up to the next word boundary.

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 realtime 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

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 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].

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 `ttyname()` 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 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.

5.5.2.4 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 2.4; 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 5.4 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 teminal 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. To ensure that the data has been transmitted entirely, make a call to tcdrain(). On return from tcdrain(), the written data will be transmitted.

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 5.4 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 5.4 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 5.4 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 i*mplementation may define* additional categories.

S. In addition to the categories (environment variables) described in the standard, the SunOS 5.4 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.

8.3.2.2 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`.

6

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, with the following exception: names with an embedded `NULL` will be truncated at the `NULL`.

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 2.4; hence, there is no implementation-specific behavior.

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 2.4 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 2.4; 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 2.4, 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.

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 2.4; 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 2.4; 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 2.4; hence, there is no implementation-specific behavior

De Jure Standards 7

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.

7

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 3.0.1 Transition Guide for SPARC Systems*–Sun Microsystems
- *SPARCompiler C 3.0.1 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*

7

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 2.4 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.

7

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