sun microsystems

Sun386i Developer's Guide

May 1988

Sun Microsystems, Inc. • Two Federal Street • Billerica, MA 01821 • (617) 667-0010

Part No: 814-1009-10
Revision A, May 1988

Credits and Copyright

Catalyst℠ is a servicemark of Sun Microsystems, Inc. SunCore®, Sun Microsystems®, Sun Workstation®, and the Sun logo are registered trademarks and Diagnostic Executive™, DOS Windows™, NFS™, PC-NFS™, SPARC™, Sun™, SunGKS™, SunSimplify™, SunOS™, SunCGI™, SunIPC™ SunLink™, SunView™, Sun Organizer™, Sun-2™, Sun-3™, Sun-4™, Sun386i™ are trademarks of Sun Microsystems, Inc.

SunINGRES™ is a trademark of Sun Microsystems, Inc. and is derived from INGRES, a product marketed by Relational Technology, Inc.

SunUNIFY™ is a trademark of Sun Microsystems, Inc. and is derived from UNIFY®, a product of Unify Corporation.

AT™, PC™, and PC/XT™ are trademarks, and IBM® is a registered trademark of International Business Machines.

Ada™ is a trademark of the Joint Program Office, U.S. Department of Defense.

DEC®, PDP–11®, VAX®, and VT® are registered trademarks of Digital Equipment Corporation.

Ethernet® is a registered trademark of Xerox Corporation.

Frame Maker™ is a trademark of Frame Technology Corporation.

Intel® is a registered trademark of Intel Corporation.

Interleaf™ is a trademark of Interleaf, Inc.

Lotus® is a registered trademark of Lotus Development Corporation.

Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.

Motorola® is a registered trademark of Motorola, Incorporated.

PostScript™ is a trademark of Adobe Systems, Inc.

UNIX® is a registered trademark and UNIX System V™ is a trademark of AT&T in the USA and other countries.

VAX® is a registered trademark of Digital Equipment Corporation.

VMEbus™ is a trademark of Motorola, Incorporated.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

NOTICE

This equipment complies with the requirements in Part 15 of the FCC rules for a class A computing device. Operation of this equipment in a residential area may cause unacceptable interference to radio and TV reception requiring the operator to take whatever steps are necessary to correct the interference.

Operation of this equipment with Class A modules and devices in a residential area is likely to cause radio frequency interference (RFI) in which case the user at his own expense will be required to take whatever measures are required to correct the interference.

Operation of this equipment with non-Sun Microsystems equipment may also cause RFI in which case the user at his own expense will be required to take whatever measures are required to correct the interference. The 80-volt DC outlet and the IEC type power outlet located on the back of the main computing unit (the system unit) supply power to the 15" Sun monochrome monitor and the expansion unit, respectively. Any other use of these outlets may cause RFI and increase the risk of fire, in which case the user at his own expense will be required to take whatever measures are required to correct the interference or repair the damage.

Revision History

Rev	Date	Comments
A	May 1988	First release of this manual.

Contents

Figures

Tables

Preface

Audience for this Manual

This manual is for developers who want to write or port applications to run on the Sun386i™ system. It assumes application programming experience and an understanding of the C programming language, but does not assume familiarity with Sun systems. This manual, in conjunction with related Sun programming manuals described in Chapter 1, provides most of the information necessary to port any application to the Sun386i system. Comparisons of Sun386i and Sun-3™ features are provided to assist programmers in porting Sun-3 applications to the Sun386i system, and to help Sun™ newcomers broaden their understanding of the capabilities and distinct features of Sun systems.

Using this Manual

This book covers three major areas:

1. Chapters 1 through 5 provide a Sun386i overview, and discuss installation of the software that you will need, and the hardware and software porting issues that you should know about.
   - Chapter 1 gives a synopsis of the Sun386i system, and points to additional documentation that you will need to write or port applications to this system.
   - Chapter 2 describes how to install the Developer's Toolkit, which includes the C compiler, assembler, link editor, and `dbx` and `dbxtool`, in addition to other development software.
   - Chapter 3 provides an overview of the porting environment from a hardware perspective, including details of portability and compatibility issues.
   - Chapter 4 describes the porting environment from a software point of view, including details of portability and compatibility issues.
   - Chapter 5 summarizes porting information presented in the two preceding chapters.
2. Chapters 6 through 10 describe the hardware and software features available on the Sun386i system.
   - Chapter 6 considers the Sun386i user interface, including the window system, graphics, file system utilities, on-screen help, ease-of-use administration, and color features.
   - Chapter 7 describes the MS-DOS™ environment, giving a brief overview of the MS-DOS window program called `dos`, and includes information on MS-DOS issues that pertain to porting.

- Chapter 8 provides an overview of AT™ bus and peripheral device issues for the Sun386i system, including a description of dynamically loadable drivers, which are new with the Sun386i system.
- Chapter 9 describes the organization of system software on the Sun386i system, and provides the steps you should follow when distributing your application software for this workstation.
- Chapter 10 presents application guidelines that you can use to help sell your product in the international marketplace. It also includes information for country distributors to localize certain aspects of the Sun386i system.

3. The book ends with a series of appendices providing additional, detailed information in areas of interest to developers.
   - Appendix A provides a more thorough overview of the entire Sun386i system.
   - Appendix B contains the Intel® 80386 assembly language definition.
   - Appendix C presents information on the Sun386i file system layout.
   - Appendix D gives details of the Common Object File Format (COFF), used by the Sun386i system.
   - Appendix E details the differences between the C language on the Sun386i system and the C language documented in *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie.
   - Appendix F describes the Sun386i implementation of C. This information should be especially useful to C programmers incorporating assembly language routines.
   - Appendix G lists `man` pages that are new for the Sun386i system, as well as those that were modified for and those that are not applicable to this system.
   - Appendix H contains character conversion tables showing how characters are mapped when you convert MS-DOS files to ISO format and ISO files to MS-DOS format.

Conventions

This manual uses the following typographic conventions:

- Prompts, system messages, file names and path names, and code examples appear `in this font`. Where differentiation is needed, SunOS™ files are shown in lowercase, and MS-DOS files are shown in uppercase.
- Commands or responses that you should enter are shown **`in this font`**.
- Menu items appear in this font. Where you are to choose a menu item to perform a task (as opposed to just a description of a menu item), **this font is used.**
- Command text shown *in italics like this* indicates a variable for which you must substitute an appropriate value. Italics are also used for notes, designated by NOTE in the left margin.
- Key names are rectangles around letters, symbols, or words that indicate actual keys to press, such as [Help].

1

Introduction

1

Introduction

This chapter presents an overview of the Sun386i workstation. It describes the key features of the system, lists application development tips, and summarizes the manuals that you'll need to port or develop software to run on this machine.

1.1. The Sun386i System at a Glance

The Sun386i system is a new, low-end workstation based on the Intel 80386 microprocessor. Although the system uses Intel architecture, the Sun386i supports the same operating system, program development tools, and software found on Sun systems that use either the Motorola MC680x0 chip or Sun SPARC architecture. Specifically, the Sun386i supports:

- SunOS operating system — a convergence of the 4.2 BSD UNIX® and UNIX System V™ operating systems — on a 386 machine
- Sun Visual/Integrated Environment for Windows (SunView™) applications, both color and monochrome
- The Sun Network File System (NFS™)
- The same types of Sun and third-party applications that run on the Sun-3 product line
- Applications written specifically for this machine

Sun386i Features

In addition to providing most of the features of other Sun systems, the Sun386i system also offers:

Integrated MS-DOS — MS-DOS is merged right into the SunOS system, and supports all MS-DOS development tools. The Sun386i system provides an excellent cross-development environment for MS-DOS. For instance, you can use the SunOS `make`(1) capability with MS-DOS compilers and linkers. You can invoke PC™ applications and commands from both SunOS and MS-DOS command processors, and they run in special MS-DOS windows. Multiple MS-DOS windows can be open and running applications concurrently, enabling you to compile, edit, and test programs at the same time. In addition, the Sun386i system supports piping between concurrent PC applications, and between SunOS and MS-DOS processes.

AT bus — The system has three AT/XT and one XT-compatible slots, user-configurable for system expansion.

Integrated mass storage — Standard configurations are available with 91 or 327 Mbyte (megabyte) hard disks and a 3 1/2-inch diskette drive. Optionally, an expansion unit can include additional disk drives as well as a tape drive for 1/4-inch streamer tapes.

Easy-to-use administration tools — A series of integrated system administration tools help users with common administrative tasks. Such tasks include adding a system to a network (these tools enable a new system to be booted and on the network within 30 minutes of unpacking), automatic creation of new user accounts, and backup.

On-screen help facility — Cursor-sensitive on-screen help is available for most SunView applications. The on-screen help facility has a programmatic interface so you can provide your users with customized, solutions-oriented help. In addition, system (kernel) messages have been rewritten into problem descriptions that less experienced users can understand.

New keyboard — The Sun386i keyboard is compatible with the Sun-3 keyboard, but provides a superset of Sun-3 and AT-style keys. In addition to the special [Help] key, the new keyboard also provides [Compose] and [Alt Graph] keys for the composition of various international characters.

Internationalized product — The system offers keytop legend sets appropriate for various West European countries, and supports 8-bit data in the kernel, Bourne Shell, Text Editor, and MS-DOS windows.

1.2. Key Development Goals

Sun Microsystems® has long recognized the importance of attracting premier software developers to its workstation product lines. With the proper set of software solutions, both the workstation and the solution packages do well. Sun wants you to participate in the success of the Sun386i workstation by helping you:

- Create portable code
- Exploit color
- Use the new on-screen help facility
- Make your applications easily installable
- Make your drivers "loadable"
- Internationalize your applications

If you need any information about third-party products that run on a Sun386i system, contact your Sun sales office and ask about the Sun Catalyst^{SM} program.

Creating Portable Code

Sun Microsystems is a multi-architecture systems supplier. To make your work easier in this environment, Sun provides common interfaces between the hardware and the operating system and between the operating system and the software tools provided for development, communications, windows, and graphics. Use the documented interfaces; in particular, use the SunView window system user interface and toolkit. This will do the most to ensure the portability of your applications within the Sun families. Chapter 6 provides a brief introduction to the window environment; refer to the *SunView Programmer's Guide* for more information.

Exploiting Color

The Sun386i system was designed to run color applications. Accordingly, the workstation:

- Provides color frame buffers and monitors as options
- Runs SunView 1.75, which includes color facilities
- Comes with documentation that describes the color programming facilities in SunView and provides detailed guidance on their use. The *Pixrect Reference Manual*, the *SunView Programmer's Guide*, and Chapter 6 of this manual provide the information you need.

Using the New On-Screen Help Facility

This facility lets you add an important ease-of-use feature to your applications. The programmatic interface described in Chapter 6 contains all of the information you need to provide help for your applications. You can use the on-screen help that comes with the system as a model.

Making Your Applications Easily Installable

You can make your applications easily installable by creating a few required files and then using the `bar(1)` command to put those files and your application on tape or diskette. When you follow the steps in Chapter 9, users can easily load your software using a new installation program that is part of the Sun386i system's simplified administration tools.

Making Your Drivers Loadable

If you are providing drivers for your applications, make them dynamically loadable so users can load your drivers without having to reconfigure and reboot the kernel. Chapter 8 includes a brief overview of loadable drivers; *Writing Device Drivers for the Sun Workstation* contains a complete description.

Internationalizing Your Applications

Chapter 10 provides information and guidelines for creating new or altering existing applications that can sell outside of the United States. The chapter contains tables to help you establish country-specific characters for display with the [Alt Graph] key, and describes the steps necessary to translate error messages. Chapter 10 also includes information about MS-DOS issues. In addition, Appendix H contains tables showing the character mapping that occurs when converting MS-DOS files to ISO format and ISO files to MS-DOS format.

Related Documentation

Four primary sets of printed documentation support the Sun386i system:

- The *Sun386i Owner's Set*, containing manuals for first-time Sun386i users whose primary interest is in running applications
- The *Sun386i Owner's Supplement Documentation Set*, containing primarily more advanced manuals that pertain to all Sun systems, including the Sun386i system
- The *Sun386i Developer's Toolkit Documentation Set*, containing manuals necessary for application development on the Sun386i and other Sun systems
- The *Sun386i Upgrade Documentation Set*, containing technical manuals specific to the Sun386i. The upgrade set is a complement to the documentation set for SunOS 4.0.

This manual is part of the *Developer's Toolkit Documentation Set*. Table 1-1 gives a synopsis of the other manuals in this set. Whether you are porting existing Sun applications to the Sun386i system or are porting applications from another environment, you'll need access to these manuals to do your work.

Table 1-1 *Developer's Toolkit Documentation Set*

Title	***Synopsis***
Sun System Services Overview	Summarizes SunOS operating system facilities. Includes information on System V compatibility, memory management, resource controls, the file system, devices, processes and interprocess communication, and network services.
PROM User's Manual	Contains information on the boot PROM, ID PROM, and EEPROM. Explains how to get the EEPROM to reconfigure the system to recognize primary terminal or console; display a particular logo; boot from a specified device; change console display size; shorten the RAM self-test at power on.
C Programmer's Guide for the Sun Workstation	Includes an introduction to C and sections on program environments, processes, signals and interrupts, I/O, memory management, and data representation.
Programming Utilities and Libraries for the Sun Workstation	Discusses C shell, Bourne shell, System V, and streams applications programming; performance analysis; Source Code Control System (`sccs`); `make`, for building and maintaining programs; `lint`, a C portability and type rules checker; `dc` and `bc` calculators; `m4` macro processor; `lex` for generating lexical analysis programs; `yacc`, a compiler compiler; and Curses screen updating facility.
Program Debugging Tools for the Sun Workstation	Describes the `dbx` symbolic debugger and `dbxtool`, and the `adb` assembly debugger; includes tutorials and examples.
Network Programming on the Sun Workstation	Provides an introduction to networks; protocol specifications; 4.3 BSD network compatibility; Remote Procedure Calls (RPCs); Network File System (NFS); System V STREAMS interface; status monitor; Remote Execution Protocol features; Yellow Pages (YP) database.

Table 1-1 *Developer's Toolkit Documentation Set (continued)*

Title	***Synopsis***
Writing Device Drivers for the Sun Workstation	Discusses development of device drivers in general and the specific routines needed. Includes loadable driver information.
SunView Programmer's Guide	Describes SunView concepts, window types, and the attributes and functions to perform tasks within this environment. Critical to the development of any window-based application.
SunView System Programmer's Guide	Provides information on specialized SunView features and the low-level window system routines that support SunView.
Pixrect Reference Manual	Describes the Pixrect graphics library, a low-level RasterOp library for writing device-independent graphics applications.
SunCGI Reference Manual	Describes SunCGI, an implementation of the ANSI Computer Graphics Interface (CGI) for development of interactive graphics applications.

2

Installing SunOS Developer's Toolkit

2

Installing SunOS Developer's Toolkit

SunOS Developer's Toolkit is one of the two primary divisions of system software on the Sun386i workstation. To have complete SunOS 4.0 functionality, you must have both Application SunOS, the other primary division of system software, and the Developer's Toolkit loaded on your system. Chapter 9 of this manual briefly describes Application SunOS; for more detailed information about it, including loading instructions, refer to *Sun386i System Setup and Maintenance.*

This chapter contains:

- A brief description of SunOS Developer's Toolkit
- Descriptions of commands to add and remove individual pieces of the Developer's Toolkit

2.1. Contents of SunOS Developer's Toolkit

SunOS Developer's Toolkit is divided into groups of related programs and files called *clusters* that enable modular loading and unloading of software to save disk space. The names and contents of these clusters are shown below.

base_devel — software development commands and utilities such as the C compiler, assembler, link editor, `dbx`(1); you must load this cluster to be able to use any of the Developer's Toolkit with the exception of the `help_guide` cluster, which does not require `base_devel`

config — System V files necessary to reconfigure the kernel such as `config`(8) and the `/usr/sys` directory

sunview_devel — SunView development libraries required for writing window-based applications

plot_devel — libraries such as `libplot.a` and `libplotbg.a` for development plotting functions

help_guide — *Help Writer's Handbook* for writing on-screen help for applications

proflibs — profiled libraries (denoted by the suffix `_p.a`) such as `libc_p.a`, `libm_p.a`, and `libcurses_p.a`

sccs — commands required by SCCS, the Source Code Control System

sysV_devel — libraries required to port System V applications, including utilities in `/usr/5bin` and `/usr/5lib` directories

For a complete list of files contained in each of these clusters, display or print `/usr/lib/load/filesizes`. This very large file includes descriptions of all Sun386i system files.

2.2. Installation Steps

Developer's Toolkit is available on tape or diskettes. To install it from either medium, use the `loadc`(1) command with the syntax `loadc` *clustername(s)*. For instance, if you want to load all Developer's Toolkit clusters, put the tape or diskette in the drive and type:

```
loadc base_devel plot_devel sccs sunview_devel
sysV_devel proflibs config help_guide
```

All clusters, with the exception of `help_guide`, require the presence of the `base_devel` cluster to work.

2.3. Loading and Unloading Clusters After Installation

You also can use the `load`(1) and `loadc`(1) commands to add individual clusters to the Sun386i system after installation. Similarly, you can remove individual clusters with `unload`(1) and `unloadc`(1). A fifth command, `cluster`(1), provides information to help you decide which clusters you might want to load. Table 2-1 below provides the syntax and descriptions for all five commands.

Table 2-1 `load`(1), `loadc`(1), `unload`(1), `unloadc`(1), *and* `cluster`(1) *Command Syntax*

Command	*Use*
`load` [*filename* ...]	Adds the cluster(s) containing the file(s) specified to the system
`loadc` [*clustername* ...]	Adds the cluster(s) specified to the system
`unload` *filename* ...	Removes the cluster(s) containing the file(s) specified
`unloadc` *clustername* ...	Removes the cluster(s) specified
`cluster` [*filename*]	Displays the name of the cluster containing the file specified

If you enter any of the commands in Table 2-1 without an argument, the system displays a summary of all Application SunOS and Developer's Toolkit clusters, including whether or not a cluster is loaded and its size.

When you invoke either the `load`(1) or `loadc`(1) command with an argument, the system:

1. Locates the cluster specified (`loadc`), or the cluster that contains the file specified (`load`)

2. Prompts you to enter the medium from which you'll be loading the cluster, and then prompts you to insert a particular diskette or tape and to confirm that you have done so
3. Checks to see if there is enough free disk space for the cluster
4. Adds the cluster to the system if there is enough space remaining
5. Displays the amount of space taken by all currently loaded clusters, and the amount of free space remaining on your system

When you use either the `unload`(1) or `unloadc`(1) commands, the system:

1. Prompts you to confirm your decision to remove a cluster
2. Displays a message stating that the cluster has been removed (if you answer **y** to the prompt), as well as the amount of space taken by all currently loaded clusters and the amount of free space remaining on your system

For more information about these commands, refer to the `load`(1), `loadc`(1), `unload`(1), `unloadc`(1), and `cluster`(1) descriptions in the *SunOS Reference Manual.*

3

Porting and Development Environment: Hardware

3

Porting and Development Environment: Hardware

This chapter considers hardware features of the Sun386i system that have implications for the porting and portability of applications. Appendix A contains a more complete Sun386i system description.

3.1. Hardware Overview

Table 3-1 below summarizes key features of the Sun386i system that collectively determine the hardware porting and development environment. For reference, the corresponding data for a Sun-3 system is also shown.

Table 3-1 *Hardware Environment Summary*

	Sun-3 System	*Sun386i System*
Main Processor	68020	80386
Math Coprocessor	68881	80387
I/O Ports	2 serial (RS-423)	Serial (RS-423, PC compatible), Parallel (PC-compatible), SCSI
Ethernet	Yes	Yes
Mass Storage	Hard disk, 1/4-inch tape	Hard disk, 3 1/2-inch diskette, optional 1/4-inch tape
Bus	32-bit VME	16-bit AT (expandable) and System bus (32-bit proprietary)
Bus Controller	Proprietary	82380
Frame Buffer	1152x900x1, 1152x900x8, or 1600x1280x1	1152x900x1, 1152x900x8, or 1024x768x8
Monitor	19-inch monochrome, 1152x900 standard; optional color/gray-scale, 15-inch, 19-inch	15-inch monochrome, 1152x900 standard; 19-inch monochrome; 14-inch (1024x768), 16-inch, 19-inch color
Keyboard	Sun-3	Sun-3/AT superset, plus Help, Compose, and Alt Graph
Mouse	Optical (100 dpi)	Optical (200 dpi)
Graphics Support	Optional GP, GB (VME)	N/A
Main Memory	4 – 16 Mbytes	4 – 16 Mbytes

The 80386 main processor is central to the Sun386i system's hardware architecture. The implementation of the SunOS operating system on this processor handles most architectural dependencies in low-level operating system internals. That is, Sun386i users and application programmers working in high-level languages should not notice differences from a Sun-3 system, for example, that are attributable solely to processor differences.

The differences that do exist lie in the area of assembly language programming, and in the somewhat less obvious area of memory organization. Assembly language programming is considered elsewhere (see Software Development Tools, starting on page 29). Memory organization—byte ordering in particular—is discussed in the following section.

The system architecture is divided into three functional areas: the CPU, frame buffer/graphics, and main memory. The rest of this chapter considers each section in detail.

3.2. CPU

The CPU includes the 80386 main processor, coprocessors, system interfaces and mass storage devices, and buses. As noted above, any effects that these components have on porting will likely relate directly to the architecture of the 80386 itself. However, the subsections below discuss the other components as well to provide contextual information for the Sun386i system.

80386 Main Processor

Common, processor-dependent issues affecting the porting of applications are byte ordering, word size, and data alignment. These are discussed in turn below.

Byte Ordering

The 80386 is a 32-bit processor. This means that all data read or written by the processor passes through 32-bit wide registers. The order in which the data—the bytes and bits—is arranged in the 80386's registers is similar to some 32-bit processors (e.g., DEC VAX®), but differs from that found in others (e.g., 680x0, SPARC, IBM® 360). The bytes in a 32-bit integer in the CPU's register, when read from address n, end up in the register as shown in Figure 3-1 below.

80386 and VAX

Byte n+3	*Byte n+2*	*Byte n+1*	*Byte n*
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00

680x0 and SPARC

Byte n	*Byte n+1*	*Byte n+2*	*Byte n+3*
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00

Figure 3-1 *Byte and Bit Ordering in the 80386, VAX, 680x0, and SPARC*

In situations where two processors with opposite byte ordering are trying to interpret the same data, the potential exists for "byte swap" problems. There are four general areas where byte swap problems might exist:

- Reading a binary data file created by a 680x0 machine on an 80386 machine and vice versa — This produces useless data due to the differing byte order. To swap the bytes and get the correct interpretation on the 80386, you must first filter the file and then read it.
- Graphics — To maintain architectural consistency with 80386 byte ordering, the monochrome frame buffer for the Sun386i system scans the bits in the direction opposite to that of 680x0-based systems. This creates a "bit flip" problem affecting the interpretation of graphics files and graphics included in source code interpreted during compilation. This issue is discussed further in Section 3.3 on the next page.
- C — Certain constructs in C that behave properly in 680x0-based architectures can cause difficulties on 80386 systems and vice versa because of byte ordering. Applications ported from such architectures will compile properly but produce incorrect results when run. The C description starting on page 33 contains more information.
- Latent bugs — Occasionally, bugs in code running on a 680x0 system will not become apparent until the code is ported to an 80386 system.

Word Size and Data Alignment

Chapter 4 also discusses the potential for certain C constructs to create difficulties relating to data alignment in memory. Bytes (8 bits), words (16 bits), and doublewords (32 bits) are the fundamental data types that the 80386 uses. The processor places no constraints on the alignment of data in memory. For example, words do not have to have even-numbered addresses, and doublewords do not have to have addresses evenly divisible by four. For better performance, the C compiler does impose such constraints, however. The resulting alignment scheme can produce problems when interpreting structures created on a 680x0-based system—see the section on C, starting on page 33, for details.

Math Coprocessor

The Sun386i system contains an 80387 floating-point coprocessor. Because this chip is standard, no compiler switch is needed to use it. The compiler always instructs the main processor to shunt floating point operations to the coprocessor.

System Interfaces and Mass Storage

The Sun386i system includes:

- An Ethernet® port
- An RS-423 port, PC compatible
- A parallel port (output only), PC compatible
- A Small Computer System Interface (SCSI) port

The SCSI interface supports up to seven devices in a daisy-chain configuration. In the standard (91 Mbyte) diskful system, one device is on the chain already, leaving six spots free for the addition of Sun devices.

In systems including an expansion unit, the SCSI interface is routed externally from the SCSI connector on the back of the CPU board to a connector on the expansion unit. A tape drive in the expansion unit makes use of the SCSI interface. This still leaves some spots open on the daisy chain for additional devices. The expansion unit also provides a second SCSI connector for your use.

Standard diskful Sun386i systems contain a 91 Mbyte hard disk and a drive supporting 3 1/2-inch diskettes (1.44 Mbyte). An optional external 327 Mbyte hard disk and a cartridge tape drive are also available.

Buses

The Sun386i system supports two primary buses under the control of an Intel 82380 32-bit integrated DMA, interrupt, and timer controller. They are:

- 16-bit AT bus
- Proprietary 32-bit System bus

The AT bus connects to four slots (1 – 4) on the backplane, the last three of which are user-configurable AT slots supporting 16-bit devices. Slot 1 is a user-configurable XT slot for an 8-bit device. Internally, the AT bus also supports the diskette and parallel port controllers. Chapter 8 provides additional information on AT bus issues.

The System bus connects to four additional slots (5 – 8) on the backplane, the last three of which can be used only for memory expansion boards. Slot 5 can support a frame buffer board.

3.3. Frame Buffer and Graphics

The frame buffer and graphics hardware form the second of the three major functional sections of the Sun386i system architecture. The frame buffer and graphics section is linked to the CPU by the System bus.

Frame Buffers

The standard frame buffer for the Sun386i system is a monochrome unit with a resolution of 1152x900 (the same as a Sun-3 frame buffer). Optionally, the Sun386i system can have a 1024x768x8 (IBM standard resolution) color frame buffer or an 1152x900x8 color unit.

If you are writing graphics applications to run on the Sun386i system, do not write directly to the frame buffers. Instead, use the SunView and the Sun graphics libraries (appropriate for many applications) or use the routines that belong to the Pixrect graphics library. The Pixrect library is a low-level package that sits on top of the device drivers. The library contains RasterOp routines that are common among all workstations and are used to access and manipulate rectangular regions of a display device in a device-independent fashion. If you develop applications using Pixrect graphics routines, you will be able to port your applications easily, even as Sun enhances the graphics capabilities for its workstations. For information about these routines and their use, refer to the *Pixrect Reference Manual*.

Dual Resolution Modes

Programs written for 1152x900 resolution may appear slightly different when run on 1024x768 systems. Screen dislocations or distortions should be minor except, perhaps, near the right and bottom screen edges where truncation could occur. For programs written for Sun-3 systems that use the standard SunView and pixrect functions without references to actual pixel counts or locations, the differences should be negligible. You should take both frame buffer sizes into consideration when writing your applications.

Byte Swapping and Bit Flipping

The byte ordering issue discussed earlier on pages 18 and 19 affects the interpretation of graphics files—font files, icon files, cursor files, and screendumps—generated under 680x0-based architectures.

On the typical 680x0 frame buffer, the bits are shifted out of the word starting at the most-significant bit, bit 15. That is, the upper-leftmost pixel on the screen is typically bit 15 of word 0 in frame buffer memory. The next pixel, scanning from left to right as on the screen, is bit 14. The pixel to the right of the first 16 pixels displayed comes from word 1, bit 15. When interpreted as integers, note where the most-significant byte (MSB) and least-significant byte (LSB) are:

```
680x0                     MSB                       LSB

word 0    |15 14 13 12 11 10  9  8|7  6  5  4  3  2  1  0|
word 1    |15 14 13 12 11 10  9  8|7  6  5  4  3  2  1  0|
 •
 •
```

For example, the integer (word) value `0x370D` in word 0 would show up on the screen with the 680x0 frame buffer as 0011011100001101.

On the typical 80386 frame buffer, the bits are shifted out of the word from the least-significant bit, bit 0, to the most, bit 15:

```
80386                    LSB                        MSB

word 0    |0  1  2  3  4  5  6  7|8  9 10 11 12 13 14 15|

word 1    |0  1  2  3  4  5  6  7|8  9 10 11 12 13 14 15|
 •
 •
```

For example, the integer (word) value `0x370D` in word 0 would show up on the screen with the 80386 frame buffer as 1011000011101100.

Note that the bytes are backward and the bits are in the opposite order. Because graphics files are usually generated as an array of words, the bytes are backward for a typical 80386 frame buffer when handling 680x0-generated files. In the case of color frame buffers, in which each pixel is represented by a byte, this creates a potential byte swap issue only, since the whole byte is used as an index into a color table. Because of these differences, transferring a graphics file from one architecture to another could result in an incorrect picture.

In the case of monochrome frame buffers, in which each pixel is represented by a single bit, scanning from right to left presents a potential bit flip problem as well. That is, the rightmost (low-order) bit of a bit field now represents the leftmost pixel on the screen.

Because of the large number of existing files using the 680x0 format, this format is the standard for describing graphics images on the 80386-based Sun386i systems as well. This eliminates the need for two sets of files in a mixed-architecture network. Consequently, if you are porting programs from other Sun systems—programs that access the frame buffer through the documented SunView and pixrect functions—byte and bit ordering is handled automatically at run time by swapping the 680x0-format images to 80386 format. Section 4.5 (starting on page 45) describes the routine that alleviates many byte-ordering problems.

Video Monitors

Available monitors for the Sun386i system include:

- 15-inch 1152x900 monochrome monitor
- 19-inch 1152x900 monochrome monitor
- 14-inch 1024x768 color monitor
- 16-inch 1152x900 color monitor
- 19-inch 1152x900 color monitor

Keyboard and Mouse

The Sun386i keyboard (Figure 3-2 below) is a superset of AT-style (84-key) and Sun-3 keyboards. It is compatible with the existing Sun-3 keyboard, although corresponding keys are not necessarily in the same position. (Keep this in mind if you're porting Sun-3 applications and your on-line or hardcopy documentation describes or shows pictures of the keyboard layout.)

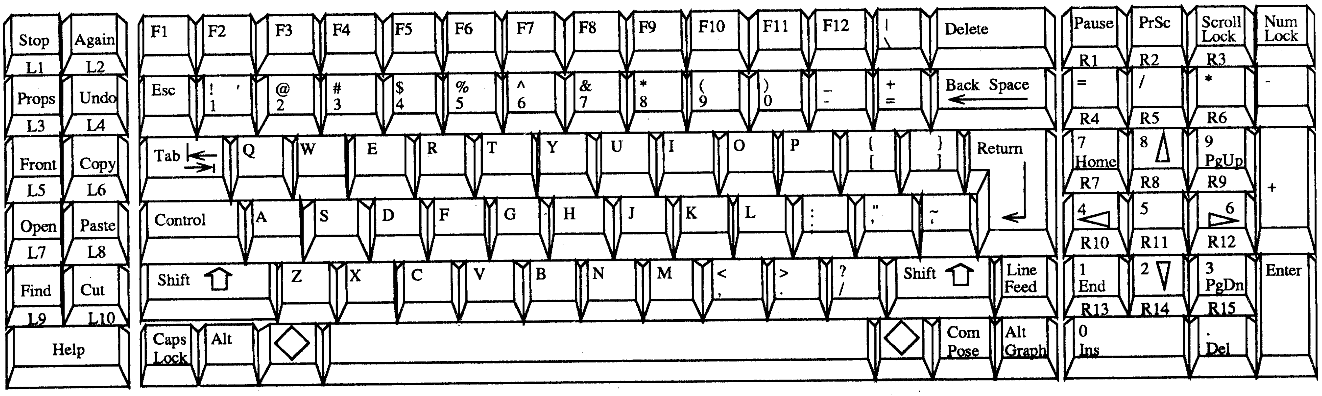

Stop L1 Again L2
Props L3 Undo L4
Front L5 Copy L6
Open L7 Paste L8
Find L9 Cut L10
Help
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 | \ Delete
Esc ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Back Space
Tab Q W E R T Y U I O P { [}] Return
Control A S D F G H J K L : ; " ' ~ `
Shift Z X C V B N M < , > . ? / Shift Line Feed
Caps Lock Alt ◇ ◇ Com Pose Alt Graph
Pause R1 PrSc R2 Scroll Lock R3 Num Lock
= R4 / R5 * R6 -
7 Home R7 8 R8 9 PgUp R9 +
4 R10 5 R11 6 R12
1 End R13 2 R14 3 PgDn R15 Enter
0 Ins . Del

Figure 3-2 *Sun386i System Keyboard: U.S. and Great Britain*

Three keys that are new with the Sun386i system are Help, Compose, and Alt Graph. The Help key implements the hardware part of a cursor-sensitive help facility. This facility is one of the ease-of-use features incorporated into the system's user interface. Chapter 6 discusses these features further.

The Compose key enables composition and use of various West European characters. Along with the keyboard shown in Figure 3-2 for the U.S. and Great Britain, additional sets of legends for other languages are available.

The Compose key enables display of many but not all additional international characters. Users can display the third character that appears on some international keycaps by using the Alt Graph key. Unlike Compose, use of Alt Graph is country specific, and is functional only if country distributors set up a keymap file, as described in Chapter 10. Chapter 10 also describes the floating accent key, available on international keyboards.

3.4. Main Memory

The Sun386i system uses Single In-line Memory Module (SIMM) boards, which use the Intel 82385 cache controller chip. SIMM boards contain sixteen slots, each of which can hold a 1 Mbyte SIMM module. The SIMM board comes equipped with 4 or 8 Mbytes of memory but you can expand it to 16 Mbytes by adding additional SIMM modules. Each system can have only one SIMM board.

4

Porting and Development Environment: Software

4

Porting and Development Environment: Software

This chapter provides a summary of Sun386i software, emphasizing Sun386i software features that have implications for the porting and portability of applications. Appendix A contains the complete Sun386i system description.

4.1. Software Overview

Table 4-1 on the following page summarizes key features of the Sun386i system that collectively determine the software porting and development environment. For reference, the table also contains corresponding data for a Sun-3 system.

The SunOS operating system and other system software on the Sun386i workstation are divided into two major sections, Application SunOS and Developer's Toolkit. (Chapter 9 describes the division of system software in more detail.) As a developer you will need both sections, which together include:

- Assembler
- C compiler
- MS-DOS 3.3
- Language, debugging, and system administration tools
- Window system and window-based applications
- Enhanced SunView tools
- Graphics packages
- Communications software

The following sections describe porting and development issues for all tools, including information on the System V Common Object File Format (COFF) used by the Sun386i system.

Table 4-1 *Software Environment Summary*

	Sun-3 System	***Sun386i System***
Core Operating System	SunOS 4.0	SunOS 4.0
Utilities, Libraries, Includes	SunOS 4.0	SunOS 4.0
MS-DOS	SunIPC™, PC-NFS	Full MS-DOS 3.3 compatibility, PC emulation, EGA support, MDA/CGA/Hercules emulation; all standard development tools; PC-NFS™
Object Code Format	`a.out`	COFF
Languages:		
Assembly	68020	80386
C	Sun 4.0 C	Sun 3.4 C
FORTRAN	Sun 2.0 FORTRAN	Sun 1.0 FORTRAN
Pascal	Sun 1.1 Pascal	Sun 1.1 Pascal
Other Language Tools	SunOS 4.0 tools	System V tools with SunOS 4.0 compatibility features
Debugging Tools	`kadb`, `adb`, `dbx`, `dbxtool`	`kadb`, `adb`, `dbx`, `dbxtool`
System Administration Tools	Standard SunOS tools	Standard SunOS tools, administration tools
Window System:		
Substrate	Pixrects	Pixrects
Toolkit	SunView 1.75	SunView 1.75
Applications	`sunview`	`sunview` plus on-screen help `(help_viewer)`, `snap`, `dos`, `coloredit`, `organizer`
Graphics	SunCore®, SunCGI™, SunGKS, PHIGS	SunCGI, SunGKS™
Communications	NFS, RPC, XDR, YP, Ethernet (TCP/IP™); all SunLink™ products	NFS, RPC, XDR, YP, Ethernet (TCP/IP); TE100, DNI, and IR
Database Management	SunINGRES™, SunUNIFY, SunSimplify™	SunUNIFY™, SunSimplify

4.2. Operating System

This section provides an introduction to the SunOS 4.0 system, an inventory of utilities, libraries, and include files, and a brief consideration of MS-DOS. If you are unfamiliar with the SunOS system, refer to *Sun System Services Overview.*

The SunOS system on the Sun386i workstation incorporates code from three sources:

- SunOS 4.0, the most recent major release of the Sun operating system
- AT&T's System V.3
- MS-DOS 3.3

The System V contributions are primarily at the level of software development tools. The core operating system is the SunOS system, which is based on a convergence of 4.2 BSD UNIX (with some 4.3 features) and System V UNIX functionality.

SunOS 4.0

If you are porting existing Sun 3.x applications, the 4.0 changes in the Sun386i core system should have little or no impact on your efforts. For example, one of the major enhancements is in the area of virtual memory management. Although this should ultimately improve performance and configurability, the change is not visible at the applications programming level. Similarly, the new file system layout (described in Appendix C) should have negligible effects, though you need to know where things are in the new scheme.

Another major SunOS 4.0 enhancement is shared library support. This capability makes for more efficient use of disk space but, again, is transparent at the applications programming level. You don't have to do anything special to use shared libraries; if a shared version of a library is available, the system uses it by default. However, you might want to make your own libraries shared by using mechanisms available with the C compiler (`cc`), assembler (`as`), and link editor (`ld`). For more information about shared library mechanisms, refer to the `man`(1) pages for the above commands and to *Sun System Services Overview.* The Utilities, Libraries, and Includes section on the next page contains a brief description of shared libraries on the Sun386i system.

Other new features, while not affecting porting, may be beneficial for new development work. A notable example in this category is the "lightweight process" capability. Lightweight processes provide a mechanism for allowing several threads of control to share the same address space. This is useful in managing asynchronous events, such as waiting for I/O operations to complete. The SunOS lightweight process library provides primitives for manipulating threads, as well as for controlling all events (interrupts and traps) on a processor. *Sun System Services Overview,* section 3L of the *SunOS Reference Manual*, and the *Change Notes and Upgrade Manual for the Sun Workstation* provide more information about lightweight processes. (The latter also provides details of changes between 3.x and 4.0 versions of the SunOS system.)

System V

If you are porting software from System V, the System V enhancements to the SunOS 4.0 system (including the STREAMS interface, partial 8-bit character support, and Base Level System V Interface Definition compatibility) also should have minimal porting impact. Section 4.3 on page 29 contains a System V compatibility overview; for a detailed look at the new System V compatibility features, refer to *Sun Systems Services Overview.*

Porting development tools from System V will be easier because the SunOS 4.0 system on the Sun386i workstation uses the Common Object File Format (COFF). COFF is discussed briefly in the Object Code Format section on page 31, and more extensively in Appendix D.

Utilities, Libraries, and Includes

Between Application SunOS and Developer's Toolkit, the SunOS system on the Sun386i workstation contains about 450 utilities for development work (too numerous to list here). The system libraries that it contains are shown below.

libc.a *
libcurses.a *
libdbm.a
libg.a
librpcsvc.a
libtermlib.a (libtermcap.a) *
libmp.a
libresolve.a
liby.a
libld.a
libkvm.a
libsunwindow.a *
libsuntool.a *
libln.a (libl.a)
lib.b
libcgi.a
libm.a
liblwp.a
libpixrect.a *
lint versions of these libraries

* Profiled version of the unshared form of this library also included (profiled versions of shared libraries not included)

Shared versions of the libpixrect.a, libsuntool.a, libsunwindow.a, libc.a, and libkvm.a libraries are part of the core system, which is shipped on the Sun386i disk. All of the above unshared libraries, including profiled versions, are part of the Developer's Toolkit. If a shared version of a library is available, the ld(1) linker dynamically links the shared library to programs specifying that library when those programs run. Shared library names have the format *library*.so.*major_rev.minor_rev;* for example, libc.so.1.0 is a shared library. If instead you want to specify use of only unshared libraries, you must include either the:

- -Bstatic option with the cc(1) command
- -Bstatic option with the ld(1) command
- Environment variable setenv LD_OPTIONS -Bstatic in your .login file

Note that you cannot use cc -Bstatic and -pic options together, since -Bstatic indicates that the program is not shared, while -pic generates shared code. The ld(1) and cc(1) man pages and *Sun System Services Overview* contain more information about shared libraries.

Integrated MS-DOS

Chapter 7 discusses MS-DOS on the Sun386i system. It is included here merely to emphasize its integration into the core system. MS-DOS permits running of text and graphics applications in DOS Windows and access from MS-DOS to the diskette drive and the SunOS file system.

4.3. Porting Overview

This section describes porting SunOS and other UNIX-based applications, as well as porting applications designed to run on other operating systems. The section also describes compatibility between SunOS 4.0 and System V systems, and discusses "negative" addresses, which you could see if you port very large programs.

SunOS and UNIX-Based Applications

You can easily port applications developed for Sun-2, Sun-3, and Sun-4 systems to the Sun386i system because the architectures are source-code compatible. Since the SunOS system is a convergence of Berkeley 4.2 BSD (with some 4.3 features) and System V systems, applications written in C, FORTRAN, or Pascal on other UNIX systems are also source-code compatible. Note, however, that the Sun386i system does not offer binary compatibility. With the exception of PC applications running in DOS Windows, you must recompile programs on the Sun386i system.

Porting most SunOS or other UNIX-based applications to the Sun386i system is a two-step procedure:

1. Copy the development tree to a Network File System (NFS) structure.
2. Recompile program modules on the Sun386i system with the `make(1)` utility.

Network Programming on the Sun Workstation describes NFS, and *Programming Utilities and Libraries for the Sun Workstation* contains information about the `make(1)` utility.

If an application does not currently use windows, consider rewriting it as a window-based program. While this requires extra work, once you have built a window-based application on the Sun386i system you can easily port it to other Sun workstations, and you can use the same model for other windowing systems. Section 6.1 on page 65 briefly describes the window system and window applications that are standard on the Sun386i system. The *SunView Programmer's Guide* and the *SunView System Programmer's Guide* contain details.

System V and Berkeley Compatibility

System V programs and commands included in the SunOS 4.0 system fall into two categories — those that are upward compatible with programs and commands in the Berkeley UNIX system, and those that are incompatible with the Berkeley UNIX system. Compatible commands are inconspicuous; they are included in regular system directories such as `/usr/bin`. System V programs that are incompatible with those in the Berkeley UNIX system reside in `/usr/5bin`. For example, the utility `/usr/5bin/stty` has an entirely different set of options from the Berkeley version, which is `/bin/stty`. You can select either version by setting your search path. Similarly, libraries and include files for compiling System V software reside in `/usr/5lib` and `/usr/5include`, respectively. To compile a program written for System V, do not use `/bin/cc`; instead, use `/usr/5bin/cc`, which will read all of the correct include files and load the correct libraries.

The SunOS 4.0 system conforms to nearly all of the requirements specified by the Base Level of the System V Interface Definition (SVID). The system includes such important System V features as record locking, named pipes, shared memory, semaphores, messages, and an emulation of the revised terminal driver. The only known SunOS 4.0 system calls that do not conform to the Base Level of the SVID are:

creat(2) and open(2V) — The creat(2) and open(2V) system calls use Berkeley semantics to assign files the group of their parent directory. System V assigns files the group of the creating process.

chown(8) — The Berkeley version of the chown(8) system call requires root privileges. On System V the owner of a file can change its ownership. This would make the Berkeley quota(1) mechanism completely unenforceable.

utime(3C) — The utime(3C) system call can't set file time stamps to the current time on NFS-mounted files, and only works on files owned by the caller. System V allows any process with write permission to a file to set that file's time stamps.

kill(1) and kill(2V) — The kill(1) and kill(2V) system calls only allow processes to send signals to other processes with the same effective user ID. The SVID specifies that a process can send a signal to processes with an effective or real user ID that matches the effective or real user ID of the sender. Root (superuser) processes can send signals to any other process.

mknod(8) — You cannot use the mknod(8) system call to create directories, as specified by the SVID; use the mkdir(1) system call instead.

fcntl(2) — The fcntl(2) system call with the F_SETFL command setting the O_NDELAY flag affects all references to the underlying file. On System V, this fcntl call affects only file descriptors associated with the same file table entry.

In addition, the current phase of System V compatibility does not fully support some System V terminal interface specifications:

character support — 5-bit and 6-bit characters are not supported.

VMIN and VTIME — VMIN (the minimum number of characters returned to the user) is always set to 1 and VTIME (the timer for short-term data transmissions) is always set to 0. In SVID, there are four possible values for VMIN and VTIME.

erase and kill characters — The initial default erase and kill characters are not # and @ respectively, but rather DEL and CTRL-U.

Applications Based on Other Operating Systems

To port applications that do not run under UNIX-based systems, you must rewrite code and then compile it on the Sun386i system. If an application already runs in a window system, porting to the Sun386i system will be somewhat simpler; the basic structure of the application, as well as the methods for processing events and displaying output, will remain generally the same. For information on writing a window-based application, refer to the *SunView Programmer's Guide* and the *SunView System Programmer's Guide*.

Porting Large Programs

For very large programs with address spaces exceeding 2 gigabytes, addresses can appear negative because signed integers are represented in two's complement notation. Mixing pointers and integers carelessly can be dangerous.

The 32nd bit, which is 1 in addresses greater than 2 gigabytes, indicates the sign of a number for a signed integer. This means that addresses between 2 and 4 gigabytes appear to be negative, if interpreted as signed integers.

In ascending order, program addresses (viewed as signed integers) start at 0 and go to 2 gigabytes (7FFFFFFF in hexadecimal). The next address in sequence is –2 gigabytes (80000000 in hexadecimal) and addresses then decrease in absolute value, approaching zero, with –1 (FFFFFFFF in hexadecimal) the largest possible address.

4.4. Software Development Tools

The software development tools described in this section are part of SunOS Developer's Toolkit. They include:

- The assembler and compilers
- Other language tools
- Debugging tools

Although other software facilities are used for program development (e.g., editors, window system tools), what unites the tools discussed here is their object file orientation. They are used either to generate object files, to generate information about object files, or to manipulate object files.

The format for the object files themselves is taken from the UNIX System V Common Object File Format, and is the focus of the following section. Appendix D contains COFF details.

Object Code Format

Use of the COFF format for object code files is key to the additional System V compatibility offered by the Sun386i system. The `man` pages and the corresponding include files listed below contain definitions of COFF data structures.

Table 4-2 `man` *Pages and Include Files Containing COFF Definitions*

man *Page*	*Include File*
`coff`(5)	`<aouthdr.h>`, `<filehdr.h>`, `<linenum.h>`, `<reloc.h>`, `<scnhdr.h>`, `<storclass.h>`, `<syms.h>`
`ldfcn`(3)	`<stdio.h>`, `<filehdr.h>`, `<ldfcn.h>`

The library `libld.a` contains functions to access and manipulate COFF object files. Table 4-3 lists the functions and briefly describes their use. The *SunOS Reference Manual* contains additional information. In most cases, the `man` page has the same name as the function; when different, Table 4-3 shows the name of the appropriate `man` page in parentheses after the description.

To use these functions, include the appropriate header files in your source code. At compile time invoke `cc`(1) or `ld`(1), with the argument `-lld` in the command line that creates the final executable module.

Table 4-3 *System V Functions for Manipulating COFF Files*

Function	*Description*
`ldaclose`	Close object file being processed (`ldclose`)
`ldahread`	Read archive header
`ldaopen`	Open object file for reading (`ldopen`)
`ldclose`	Close object file being processed
`ldfhread`	Read file header of object file being processed
`ldgetname`	Retrieve the name of an object file symbol table entry
`ldlinit`	Prepare object file for reading line number entries via `ldlitem` (`ldlread`)
`ldlitem`	Read line number entry from object file after `ldlinit` (`ldlread`)
`ldlread`	Read line number entry from object file
`ldlseek`	Seek to the line number entries of the object file being processed
`ldnlseek`	Seek to the line number entries of the object file being processed given the name of a section (`ldlseek`)
`ldnrseek`	Seek to the relocation entries of the object file being processed given the name of a section (`ldrseek`)
`ldnshread`	Read section header of the named section of the object file being processed (`ldshread`)
`ldnsseek`	Seek to the section of the object file being processed given the name of the section (`ldsseek`)
`ldohseek`	Seek to the optional file header of the object file being processed
`ldopen`	Open object file for reading
`ldrseek`	Seek to the relocation entries of the object file being processed
`ldshread`	Read section header of the object file being processed
`ldsseek`	Seek to the section of the object file being processed
`ldtbindex`	Return the long index of the symbol table entry at the current position of the object file being processed
`ldtbread`	Read a specific symbol table entry of the object file being processed
`ldtbseek`	Seek to the symbol table of the object file being processed
`sgetl`	Access long integer data in machine-independent fashion (`sputl`)
`sputl`	Translate a long integer into machine-independent format

External Symbol Representation

The Sun386i system follows the COFF convention of not prepending underscores to external symbols. On other Sun systems, external symbols require a leading underscore character. Be aware of this difference, particularly when using the `libc nlist` function to look up symbols in an object module symbol table. For instance, on a Sun386i system you would search for `proctab` instead of `_proctab`.

Assembly Language

The assembler, `as`(1), is a System V import as is the assembler definition for the 80386. Appendix B contains information on the 80386 assembler; `as` also has its own `man`(1) page. If you plan to do assembly language programming, you should probably also have a copy of Intel's *80386 Programmer's Reference Manual.* For information on assembly language calling conventions from C, refer to Appendix F.

NOTE *The 80386 assembler differs from 8086 assemblers with which you might be familiar. For instance, the Sun386i 80386 syntax for the* `mov` *instruction is* `mov source destination`. *Many 8086 assemblers place the destination variable before the source. Be careful of such differences when using the 80386 assembler.*

C

The Sun386i system's C compiler, `cc`(1), has most of the functionality and interface of C in the SunOS 3.4 system on other Sun machines. The C run-time start-up code is in the file `/usr/lib/crt0.o`.

Features that should not affect porting but which the Sun386i system C compiler does not have are:

- The global optimizer
- The `-a`, `-align`, *float_option*, `-ffpa`, `-fsky`, `-vpa`, and `-J` options

C documentation consists of:

- The `cc`(1) `man` page in the *SunOS Reference Manual*
- The *C Programmer's Guide for the Sun Workstation*
- A description of the operational characteristics of C on the 80386 under System V in Appendix F of this manual
- A list of the differences between C on the Sun386i system and the C language documented in *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie in Appendix E

If you want to use the `dbx`(1) debugger, you must use the C compiler's `-g` option. Appendix D in this manual provides more information about the COFF entries produced by the `-g` option. *Program Debugging Tools for the Sun Workstation* describes `dbx`(1), as well as the `adb`(1) and `dbxtool`(1) debuggers. Additionally, the following sections provide guidance on some porting issues.

Bit Shifting

When porting code to the Sun386i system, you could run into problems because of a difference in how bit shifting is handled. The maximum shift count for the Sun386i system is 31, unlike most machines which allow a higher shift count (even though a count above 32 is meaningless). The limit of 31 reduces maximum execution time.

If the instruction `i = y << j` is executed on a Sun386i system and the value of `j` is greater than 31, only the *lower five bits* of `j` are used for the count. This means that a shift of 32 is equivalent to a shift of zero bits. To avoid this problem, you must search through your code and check all shift operations. For each operation such as *value* << = *count;* where *count* could be greater than 31, insert the following code:

```
if (count > 31)
   value = 0;
else
   value <<= count;
```

Byte Ordering

Section 3.2 on page 18 discusses byte order differences between the 80386 architecture and some other architectures, including 680x0-based Sun systems. Certain constructs in C can get you into trouble when porting from 680x0-based architectures to the Sun386i system. However, these problems should be minimal. Three known areas that can cause difficulty are:

- Using `strlen()` and `malloc()` improperly
- Message passing over the network
- Using casts improperly

strlen( ) and malloc( )

C programmers commonly allocate memory for a character string and copy the string into the allocated space:

```
char *strptr;
char string[LEN];

strptr = (char *)malloc(strlen(string) + 1);
strcpy(strptr,string);
```

It is also easy, however, for a programmer to forget to put the `+ 1` in the `malloc()` call to account for the null termination:

```
strptr = (char *)malloc(strlen(string));
strcpy(strptr,string);
```

Now, `malloc()` memory looks something like:

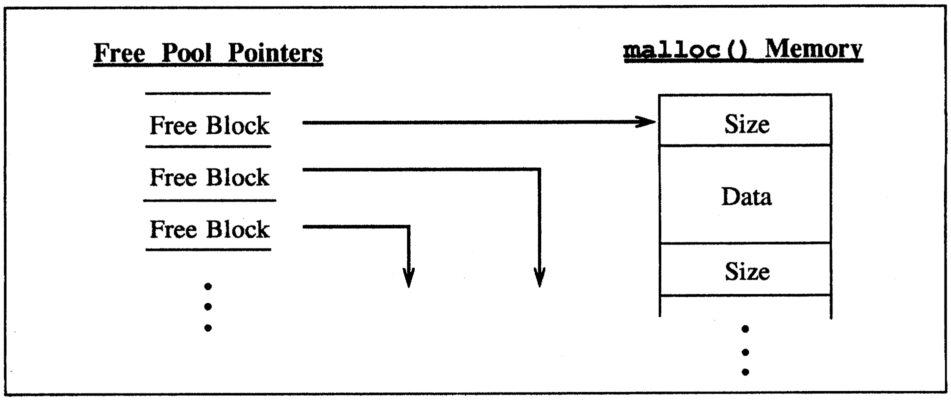


Figure 4-1 *Conceptual Representation of* `malloc()` *Memory*

If the following two conditions hold:

1. The + 1 is left out of the call, and
2. The length of the string returned by strlen() is an exact multiple of four (doubleword)

then, when the string is copied into the allocated space, the termination character (\0) will be written into the first byte of the next word. But the next word contains the size of the next piece of heap memory.

On the 680x0, this does not usually pose a problem because the byte written is the high-order byte of the size, typically zero already. For this byte to be nonzero, the size of the next piece of allocated memory would have to be greater than 16 Mbytes. On the 80386, however, the byte that gets zeroed is the low-order byte of the size—a serious problem that might surface only after porting to a Sun386i system. Therefore, be sure to include +1 in malloc() calls.

Message Passing Over the Network

Passing messages over the network can create problems because network byte order is reversed from byte order on the Sun386i system. This is only a problem if, in addition to ASCII, your messages include some binary data. You can preclude problems of this nature by using either:

- The four host-to-network and network-to-host conversion functions documented in the byteorder(3N) man page, for porting existing applications that do not use floating-point data (integer only), or
- XDR (eXternal Data Representation) functions, primarily for new applications, and for those using floating-point and integer data

The first method, using the byteorder functions, is more straightforward. However, to ensure that new applications will run on any architecture, you should use the XDR file format. Refer to Section 4.6, Data Format Issues, on page 47.

Casts

Using casts can be tricky since character-to-integer and integer-to-integer conversions between 680x0 and Sun386i systems produce different results. For instance,

```
f()
{
    int a;
    unsigned char b[4];
    b[0]= ...;  b[3]= ...
    a=*(int*) b;
    return a;
}
```

gives different answers on 680x0 and Sun386i systems. If you have a 680x0-format binary data file with integer fields, read the integer fields into a character buffer on the Sun386i system, and then cast them into integers, the result will be reversed from the result on a 680x0 system. The same is true when going from Sun386i to 680x0 systems. There are a few workarounds for this problem:

- Use eXternal Data Representation (XDR) format (see Data Format Issues on page 47 for more information).
- Read one byte at a time, shift it, and add the new byte.

Using the example of a 680x0-format binary data file with integer fields, after reading a field into the character array b, instead of saying:

```
a=*(int*) b;
```

as shown above, one alternative is to use code similar to what's shown below:

```
a=(b[0]<<24)+(b[1]<<16)+(b[2]<<8)+b[3];
```

Data Representations

On the Sun386i system, the least-significant bit of a data element is always in the lowest-numbered (rightmost) byte. This section describes `integer`, `float`, and `double` representations on the Sun386i system.

Integer Representations

Representation of `short` *on the Sun386i System*

Bits	***Address***
8 – 15	n+1
0 – 7	n

Representation of `int` *and* `long` *on the Sun386i System*

Bits	***Address***
24 – 31	n+3
16 – 23	n+2
8 – 15	n+1
0 – 7	n

Bits	***Address***
16 – 31	n+2
0 – 15	n

Representation of `double` *on the Sun386i System*

Bits	***Address***
32 – 63	n+4
0 – 31	n

`float` and `double` Representations

`float` and `double` data elements are represented according to the ANSI-IEEE 754-1985 standard. The following tables describe this representation.

`float` *Representation on the Sun386i System*

31	30 23	22 0
s	e	f

`float` *Format on the Sun386i System*

s = sign (1)
e = biased exponent (8)
f = fraction (23)	

Normalized number (0<e<255):	$(-1)^s \times 2^{e-127} \times 1.f$
Subnormal number (e=0, f!=0):	$(-1)^s \times 2^{e-126} \times 0.f$
Zero (e=0, f=0):	$(-1)^s \times 0$
Signaling NaN:	s=u; e=255 (max); f=.0uuu-uu (at least one bit must be nonzero)
Quiet NaN:	s=u; e=255 (max); f=.1uuu-uu
Infinity:	s=u; e=255 (max); f=.0000-00 (all zeroes)

`double` *(high) Representation on the Sun386i System*

63	62 55	54 32
s	e	f-msb

`double` *(low) Representation on the Sun386i System*

31 0
f-lsb

`double` *Format on the Sun386i System*

s = sign (1)
e = biased exponent (11)
f = (f-msb, f-lsb) = fraction (52)	

Normalized number (0<e<2047):	$(-1)^s \times 2^{e-1023} \times 1.f$
Subnormal number (e=0, f!=0):	$(-1)^s \times 2^{e-1022} \times 0.f$
Zero (e=0, f=0):	$(-1)^s \times 0$
Signaling NaN:	s=u; e=2047 (max); f=.0uuu-uu (at least one bit must be nonzero)
Quiet NaN:	s=u; e=2047 (max); f=.1uuu-uu
Infinity:	s=u; e=2047 (max); f=.0000-00 (all zeroes)

Data Alignment

The Sun386i system imposes no constraints on the alignment of data in memory, but the C compiler does: bytes on byte boundaries, words (16 bits) on word boundaries, and doublewords (32 bits) on doubleword boundaries. Such a scheme is referred to as aligning data on natural boundaries. This enhances performance on Sun386i systems. On the 680x0 systems, however, characters are aligned on byte boundaries and everything else, regardless of size, is aligned on word (16-bit) boundaries. Thus, if you create a structure such as

```
struct  {
          char a;
          int b;
        }y;
```

its arrangement in memory on a 680x0-based versus a 80386-based system will look something like Figure 4-2.

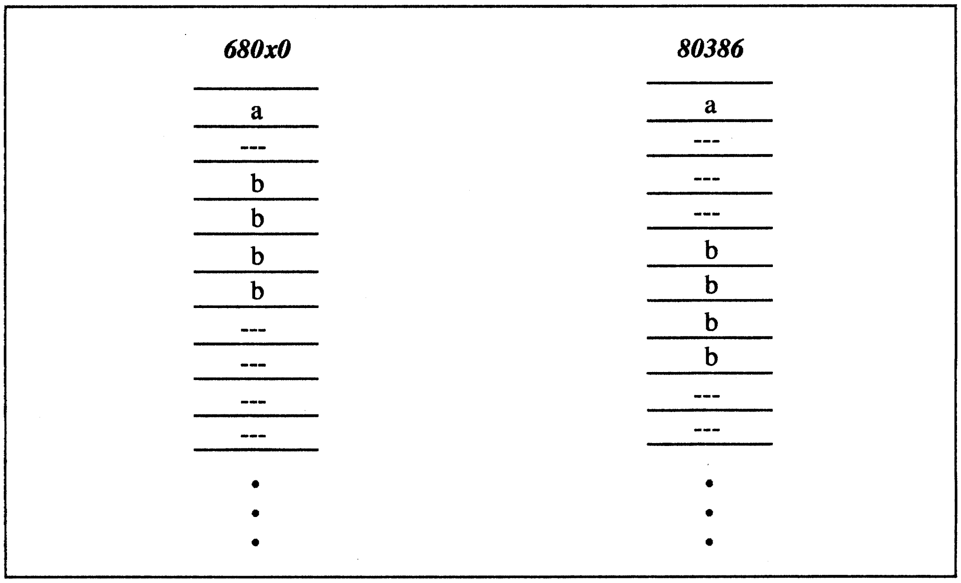


Figure 4-2 *Conceptual Representation of a C Structure in 680x0 versus 80386 Memory*

The C compiler on 680x0-based machines aligns the four-byte integer on the next word boundary after the character. On the Sun386i (80386-based) system, however, the integer is aligned on the next doubleword boundary. Consequently, reading data created by one type of system from the other type, whether from a disk or over the network, produces errors owing to the different alignment schemes.

Other Porting Issues

The following are other potential sources of C-related porting problems, particularly if you are porting from Sun 680x0-based systems:

- Symbols predefined by the C preprocessor (cpp)
- Casting a structure to a scalar value
- asm function declarations

Symbols Predefined by cpp(1)

The C preprocessor, cpp(1), is derived from the SunOS 4.0 system, not System V. It predefines several symbols: i386, sun, and unix. It's a good idea to use these symbols in #ifdef statements as follows:

- i386 — 80386 processor-specific
- sun — Sun workstation-specific
- unix — UNIX operating system-specific

The cpp preprocessor does **not** predefine the following symbols: mc68000, m68k, mc68010, mc68020, sun386.

Casting a Structure to a Scalar Value

The C compiler does not allow you to cast a structure to a scalar value; therefore, the following code does not work with the Sun386i C compiler:

```
/*
*This example causes an error with the casting of
*a structure to a pointer, even though the struc-
*ture is the same size as the pointer.
*/
typedef char * Textsw_opaque;
typedef Textsw_opaque Textsw_mark;
typedef struct ev_mark_object {
      unsigned    move_at_insert : 1;
      unsigned    spare : 15;
      unsigned    id : 16;
} Ev_mark_object;

main()
{
      Ev_mark_object    x;
      Textsw_mark       y;

      y = (Textsw_mark)x;     /*This is the   */
}                             /*problem stmnt.*/
```

A workaround for this is:

```
/*
*Here is a workaround for the above. The standard
*C definition only allows structs and unions to be
*cast to objects of the same type.
*/
typedef char * Textsw_opaque;
typedef Textsw_opaque Textsw_mark;
typedef struct ev_mark_object {
      unsigned    move_at_insert : 1;
      unsigned    spare : 15;
      unsigned    id : 16;
} Ev_mark_object;
typedef Textsw_mark * tsmPtr;

main()
{
      Ev_mark_object    x;
      Textsw_mark       y;
      tsmPtr            z;

      /*The next, 2-instruction sequence works*/
      z = (tsmPtr)&x;
      y = *z;
      /* OR */
      /*This sequence also works*/
      y = *((tsmPtr)&x);
      /* OR */
      /*So does this*/
      y = *((Textsw_mark *)&x);
}
```

asm Function Declarations

The C compiler does not allow `asm` function declarations. However, you can embed assembler statements in C code (called *inlining*) by using the `asm` mechanism:

```
asm ( " ... assembler statement ...");
```

You also can reference C variables and labels with inlining. In the following example, the integer `i` is declared in a C program. The `asm` statement references that integer by name.

```
int i;
.
.
.
asm ("movl  i,%eax");
```

FORTRAN

The Sun386i system can run version 1.0 of FORTRAN 77. FORTRAN is an unbundled product, which you must order separately. In addition, you must have SunOS Developer's Toolkit to run FORTRAN. If you have FORTRAN, the compiler is located in `/usr/bin` and you can access it with the `f77`(1) command. The FORTRAN library, `libmf` (the machine-independent math library), the parser, and the optimizer are in `/usr/lib`. Features of the FORTRAN compiler on the Sun386i system that may differ from other versions you have used are shown on the next page.

Options

f77 compiler options include run-time checking of array subscripts, flagging of undeclared variables, and flagging of nonANSI statements.

Extensions

f77 supports the following extensions (which, if used, could make subsequent porting more difficult):

- Recursive function and subroutine calls
- VAX/VMS® extensions, which enable you to convert FORTRAN source code written for VAX systems to run on the Sun386i system
- Automatically allocated local variables, bit fields, and Boolean operators
- Relaxed input format
- POINTER data type for supercomputer compatibility
- 16-bit integer and double-precision complex data types

The f77(1) man page and the *FORTRAN Programmer's Guide* contain additional information about the language.

Pascal

The Sun version of Pascal 1.1 runs on the Sun386i system. The compiler, pc(1) is an enhanced version of the ISO Pascal compiler, which produces significantly improved assembly code. Features of Sun Pascal that might differ from other versions of Pascal that you have used are:

Options

You can disable run-time checking of subrange and array bounds; you also can flag nonstandard uses.

Violations

The Sun compiler requires that operands have identical types, not just compatible ones.

Extensions

pc supports the following extensions (which, if used, could make subsequent porting more difficult):

- Separate compilation support
- Identifiers declared as statement labels
- Lexical extensions, including _ and $
- Public and private restrictions to the current compilation unit
- Boolean operators for defining conditional expression evaluation
- getfile function
- Access to SunOS system calls

As with FORTRAN, Pascal is unbundled so you must order it separately, and you must have SunOS Developer's Toolkit to run it. The *Pascal Programmer's Guide* and the pc(1) man page provide additional information.

Other Language Tools

Language tools discussed so far include:

as(1) — Sun386i assembler

cc(1) — C language compiler

cpp(1) — C language preprocessor

f77(1) — FORTRAN language compiler

pc(1) — Pascal language compiler

Additional Sun386i tools are listed and briefly described below; the *SunOS Reference Manual* contains complete documentation for each one. Note that both the dis(1) and objdump(1) commands are new with the Sun386i system and have no previous SunOS equivalents.

ar(1) — archiver and library maintainer

dis(1) — object code disassembler

objdump(1) — object code dumper

ld(1) — link editor

lorder(1) — find ordering relation for an object library (not needed on the Sun386i system; see the following section)

nm(1) — print name list of an object file

ranlib(1) — convert archives to random libraries (not needed on the Sun386i system; see the following section)

size(1) — print text, data, and bss (uninitialized data) section sizes of object files

strip(1) — remove symbol and line number information from an object file

lint(1V) — C program checker

prof(1) — display profile data

gprof(1) — display call-graph profile data

tcov(1), a SunOS tool that constructs test coverage analysis, is not included with the Sun386i system.

ranlib and lorder Notes

On Sun386i systems, the ar archiver and library maintainer creates and maintains a table of contents (similar to ranlib) for the ld linker. ar also makes sure that an archive's table of contents is always in sync with the archive. Therefore, you don't have to use ranlib on the Sun386i system to perform these functions. However, for compatibility with ranlib on other systems, you can use ranlib on the Sun386i system to reconstruct an archive's table of contents; if you do, ranlib calls ar with the necessary options. (For example, if you use strip to remove line numbers from the table of contents, ld will not accept the archive until you rebuild the table of contents with either ranlib or ar -ts.)

`lorder` is commonly used to put archive entries in correct order for one-pass linking. Since on the Sun386i system `ar` maintains a table of contents, the order of files is never important. `lorder` is still useful for understanding the linkage between modules in a library, however.

`lorder` and `ranlib` are part of the Sun386i system for `make` file compatibility and for compatibility between these functions on Sun386i and on other systems. Although existing uses of `lorder` and `ranlib` work on the Sun386i, you can remove them without causing problems.

Debugging Tools

Completing the software developer's toolbox are:

- `adb`(1) — assembly language debugger
- `dbx`(1) — source-level debugger
- `dbxtool`(1) — window- and mouse-based source-level debugger
- `trace`(1) — command for tracing system calls that a process makes (new with the SunOS 4.0 system)

C, FORTRAN, and Pascal compilers produce uniform object code and uniform symbol tables for use by common debugging tools such as `adb`(1) and `dbx`(1). This approach has three major advantages:

- Cross-callability — You can write code in one source language that calls modules or libraries written in another language. For example, C programs can call FORTRAN numerical libraries, and FORTRAN programs can make use of C routines for system interaction.
- Ease of migration — As new hardware technology emerges, Sun can port its compilers and make them available for the new processor by writing one code generator and peephole optimizer. You quickly benefit from the availability of new hardware with compatible software tools.
- Ease of implementing new compilers and tools — Implementing a new compiler implies only a new front-end to the code generator. It's easier to integrate software tools into the new language.

`adb` and `dbx` are briefly described below. For more details, as well as a description of `dbxtool`, refer to *Debugging Tools for the Sun Workstation.* For more information about `trace`, refer to the `trace`(1) `man` page.

`adb`

Debugging Tools for the Sun Workstation contains a complete description of the `adb` debugger, including all Sun386i information. This section lists `adb` commands that are included in the Sun386i version of the debugger, but not in versions that run on other Sun systems.

`:A` Attaches process *addr.*

`:R` Releases (detaches) the current process.

`:a` Sets a data breakpoint for when *addr* is accessed (read or written).

`:w` Sets a data breakpoint for when *addr* is written.

`$l` Sets the length in bytes (1, 2, or 4) of the object referenced by `:a` and `:w`. The default is 1.

`:z` Deletes all breakpoints.

`:e` Like `:s`, but also steps over routine calls (instead of into them).

`:u` Continues uplevel, stopping after the current routine has returned. Do not give this command as the first or second instruction in a routine.

You can precede the next five commands with either the / or ? modifier, as described in *Debugging Tools for the Sun Workstation*.

- `B` Similar to the `b` command, but prints in the current radix (default is hex).
- `h` Similar to the `x` command, but prints byte swapped.
- `H` Similar to the `X` command, but prints byte swapped.
- `M` Similar to the `i` command, but also prints machine code.
- `v` Similar to the `w` and `W` commands, except it works on a single byte.

dbx

Debugging Tools for the Sun Workstation contains a complete description of the `dbx` debugger, including all Sun386i information. This section lists the `dbx` registers for the Sun386i system.

`$ss`	Stack segment register
`$eflags`	Flags
`$cs`	Code segment register
`$eip`	Instruction pointer
`$eax`	General register
`$ecx`	General register
`$edx`	General register
`$ebx`	General register
`$esp`	Stack pointer
`$ebp`	Frame pointer
`$esi`	Source index register
`$edi`	Destination index register
`$ds`	Data segment register
`$es`	Alternate data segment register
`$fs`	Alternate data segment register
`$gs`	Alternate data segment register

On the Sun386i system, for example, to print the contents of the data and address registers in hex, type **`&$eax/16X`** or **`&$eax,&$edi/X`**. To print the contents of register `eax`, type **`print $eax`**.

You can also access parts of the 80386 registers. Specifically, the lower halves (16 bits) of these registers have separate names, as follows:

`$ax`	General register
`$cx`	General register
`$dx`	General register
`$bx`	General register
`$sp`	Stack pointer
`$bp`	Frame pointer
`$si`	Source index register
`$di`	Destination index register
`$ip`	Instruction pointer, lower 16 bits
`$flags`	Flags, lower 16 bits

Furthermore, the first four of these 16-bit registers each can be split into two 8-bit parts, as follows:

`$al`	Lower (right) half of register `$ax`
`$ah`	Higher (left) half of register `$ax`
`$cl`	Lower (right) half of register `$cx`
`$ch`	Higher (left) half of register `$cx`
`$dl`	Lower (right) half of register `$dx`
`$dh`	Higher (left) half of register `$dx`
`$bl`	Lower (right) half of register `$bx`
`$bh`	Higher (left) half of register `$bx`

The registers for the 80387 include the following:

`$fctrl`	80387 control register
`$fstat`	80387 status register
`$ftag`	80387 tag register
`$fip`	80387 instruction pointer offset
`$fcs`	80387 code segment selector
`$fopoff`	80387 operand pointer offset
`$fopsel`	80387 operand pointer selector
`$st0-$st7`	80387 data registers

4.5. Window System and Graphics Support

The SunView system supports interactive, graphics-based applications running in windows. When writing new window applications for the Sun386i system, be sure to use the programming facilities provided by SunView (described briefly in Chapter 6, and more fully in the *SunView Programmer's Guide*) rather than communicating directly with the hardware.

Similarly, if your applications must access low-level graphics facilities, use Pixrect library routines instead of writing directly to the frame buffers. If you develop applications using Pixrect routines, you will be able to port your applications easily, since the routines are common among all Sun workstations. The *Pixrect Reference Manual* contains the information you need.

If you are porting existing Sun graphics applications to the Sun386i system, the byte-ordering problem discussed in Chapter 3 affects you. If you use the SunView macro `DEFINE_ICON_FROM_IMAGE` or the `mpr_static` routine on any image generated by `iconedit(1)`, then the Sun386i system automatically handles the byte-order issue for you. (The *SunView Programmer's Guide* describes both the macro and the routine.) However, if you are working with image data from a device that you have added to a system (such as a laser scanner), you will have to correct the byte-order problem yourself. Depending on how the particular driver reads data into memory, you will have to either call `pr_flip`, a pixrect routine that alleviates this problem (outlined on the following page), or write your own routine to process this data. For a complete discussion of the `pr_flip` routine, refer to the *Pixrect Reference Manual.*

pr_flip Overview

The pr_flip pixrect routine solves the byte-ordering problem that occurs when 680x0-format graphics execute on the Sun386i system. pr_flip provides:

- Low impact on existing applications — you can port existing graphics tools and applications to the new architecture transparently
- A single format for data files across the network — font files, rasterfiles, and files created by iconedit(1) are the same across architectures
- Low impact on programmers — straightforward implementation that enables easy modification of existing code, if necessary

pr_flip handles the byte-ordering problem by flipping bits in memory pixrects. The pr_data field of a display pixrect points to memory pixrect data, as shown below.

```
typedef struct pixrect {
        struct  pixrectops *pr_ops;
        struct  pr_size pr_size;
        int     pr_depth;
        caddr_t pr_data;        /*pointer to mpr*/
} Pixrect;
```

The structure referenced by pr_data is:

```
struct mpr_data {
        int     md_linebytes;
        short   *md_image;
        struct  pr_pos md_offset;
        short   md_primary;
        short   md_flags;       /*flag bits*/
     };
```

To control the operation of pr_flip on memory pixrects, the md_flags word has two new flags, MP_I386 and MP_STATIC. If the MP_I386 flag is set (TRUE) in the md_flags word, the pixrect in question is already in 80386 display format, that is, pr_flip has already operated on it. If the MP_STATIC flag is set (TRUE) in the md_flags word, the pixrect in question is a static pixrect, meaning that the image data was defined at compile time by DEFINE_ICON_FROM_IMAGE or mpr_static.

Refer to the *Pixrect Reference Manual* for more detailed information about pixrects and the pr_flip routine.

Some Pixrect Pointers for the Sun386i System

The following list provides some suggestions and considerations to keep in mind when dealing with pixrects:

- Check code that draws manually into a pixrect; you might have to modify such code before it will port properly. The type of modification required depends on the particulars of the drawing operation.
- Perform manual operations (those not involving libpixrect routines) on a pixrect **before** converting the pixrect to 80386 format.

- Two pixrect structures cannot share the same image data file; instead of using two pointers to the same file, make a copy of the file. Then make sure each pixrect structure points to a unique data file name.
- `mem_create` creates an 80386-format pixrect on Sun386i systems.
- `mem_point` does **not** set the `MP_I386` flag. The pixrect is considered **not** flipped.
- To create an icon, use `mem_point` to make a pixrect connected to an existing static image or an image that you have created on the fly. You can use `DEFINE_ICON_FROM_IMAGE` or `mpr_static` to create static icons. All static icons are initially created in 680x0 format. The system converts static icons to 80386 format as soon as they are involved in a raster operation.

4.6. Data Format Issues

You should try to ensure that your applications can run on multiple Sun systems over a network. This section briefly describes how to use a standard data format to enable existing and new applications to run on different architectures.

Existing Applications

Most existing software saves data on disk either by:

- Converting data to an ASCII text format, making the software portable but slower in terms of I/O. Files may be larger, and partial updates of files are more difficult.
- Issuing write system calls (`write`(2) or `fwrite`(3) in the SunOS system) to save the data in binary format. This method is faster and allows partial file updates, but it is not portable unless you do one of the following two things:
  - Convert the software to a standard external file representation (preferably XDR), as described in the next section. This will probably require some program redesign, but it is the better solution.
  - Use the existing file format (using the current native byte order and floating-point representation) as the standard format. No program redesign is needed, but you could run into byte-order problems and will probably have to write floating-point and other conversion routines.

New Applications

The Sun XDR (eXternal Data Representation) standard lets you define data formats that enable your applications to run on machines that have different architectures. XDR format, which includes standard data representations, is the same format that all RPC (Remote Procedure Call) communications protocols use. The main difference between using XDR for communications versus as a file format is that file formats are typically geared toward random access (using fixed-size data), while communications formats are always stream based (allowing variable-length representations for efficiency).

If you follow the steps below, application data files will be portable between architectures without application source code changes.

1. Read the chapter on XDR and `rpcgen`(1) in *Network Programming on the Sun Workstation.* The discussion includes definitions for most standard atomic data types, with the exception of bitfields and bitmaps. If your application uses either of these data types, you will have to define your own XDR routines.

2. Create rpcgen(1) definitions of the application data file formats. If your application requires random access to data files, you cannot use variable-length data structures.
3. Submit the definitions to rpcgen(1), which will supply two files that you must use: one containing the XDR routines and another containing the header file generated for the data structures. Do not modify either of these files.
4. Use the data structures from the header file for your in-memory operations; use the XDR routines to read and write data files in the standard format that you've used. (You'll still have to set up XDR handles and perform open, close, and file-seeking operations yourself).

4.7. Optimizing Code

You can use a number of methods to increase code efficiency and decrease runtime. This description provides guidelines and examples for doing so, including the sections:

- General Principles
- Using Registers
- Writing Linear Code
- Replacing Complex Operations
- Evaluating Conditions
- Generating String Instructions
- Improving Loop Efficiency
- Using Assembler Code

General Principles

Only a small part of most programs warrant optimization. In most cases, from 4 to 20% of a program is responsible for 90% of its execution time. Try to determine which part of the code you should work on by scanning the program, checking the input (depending on program size), and using various profiling techniques such as prof(1), gprof(1), and inserting counters.

Optimize for a specific architecture, but realize that this decreases portability. Code that is highly optimized for one compiler and machine will usually be less efficient when used with other compilers and machines.

Improved performance often requires the use of more space. You must decide if increased storage requirements are worth the improvement in speed. Removing unused variables can reduce code size slightly.

Generally, the maintainability of code is inversely proportional to the amount of optimization performed. **Include comments when you optimize**, to enable others to understand and follow through on your work.

Changing the algorithm used can increase efficiency. Analyze the code to determine the best method to do the job. This description assumes that you are satisfied with the algorithm chosen, and that you want to improve the code used.

Using Registers

The 80386 does not have many general-purpose registers, and some of them are unavailable because they have special meaning. In addition, some registers can hold 8, 16, or 32 bits, while others can hold only 16 or 32 bits. The general-purpose registers used in C programs follow. These are 32-bit register names, but the description applies to smaller registers as well (for example, %eax also applies to %ax, %al, and %ah). For a complete list of 80386 registers, refer to the dbx section on pages 44-45 of this chapter.

Table 4-4 *Registers Used in C Programs*

Register	***Purpose***
%eax	General register (special meaning for certain conversion instructions)
%ecx	General register (special meaning for certain string instructions)
%edx	General register (special meaning for certain conversion instructions)
%ebx	General register, used only for register variables (unless cc -pic is used—see below); supports characters
%esi	Can hold a register variable or the source pointer for string instructions; does not support characters
%edi	Can hold a register variable or the destination pointer for string instructions; does not support characters
%esp	Stack pointer
%ebp	Local frame pointer

The -pic option produces position-independent code (PIC) for dynamically linked objects. While PIC code can be shared more easily, thereby using system resources more efficiently, it executes more slowly than code compiled without the -pic option. Using -pic also means that you can use only two register variables, %esi and %edi. If you use -pic, you cannot also use -Bstatic on the same program; -Bstatic indicates that the program is not shared, while -pic generates shared code. For more information about PIC, refer to *Sun System Services Overview* and the cc(1) man page.

Note that C programs do not use segment registers. This is because the 80386 provides ample memory space. Also note that unlike VAX and 68000 machines, on the Sun386i system only three registers are available for use as register variables: %ebx, %esi, and %edi. Therefore, **use register variables carefully**. Determining what belongs in a register can provide a greater performance boost than any other technique mentioned. Another difference between registers on 80386 and 68000 machines is that the 68000 can use the MOVEM instruction to move multiple registers to and from memory. Since the 80386 has no such instruction, two move instructions are required (to save and restore a register) every time the function containing the register variable is called.

The C compiler assigns variables to registers as it encounters them in the code. In the following example, a, b, and c are placed in registers %edi, %esi, and %ebx, respectively.

```
boo() {
        register int a,b,c,d,e;
        boo statements
}
```

Of the three registers available for register variables, only `%ebx` can support characters. Therefore, only one character register variable can be declared. Any others are placed in memory. In the following example, variables `a`, `c`, and `d` are placed in registers.

```
boo() {
        register char a,b;
        register int c,d;
        boo statements
}
```

In many cases, a function could have two or more heavily used variables. If the use of these variables is disjoint, you can use one register variable and change the code so that multiple uses can share a common register variable. However, there can never be more than three variables in registers in any single scope.

Consider the following code:

```
boo() {
        register int a,b,c;
        int d;

        first use of a
        statements that do not use d
        last use of a

        first use of d
        statements
        last use of d
}
```

The code below shows how to use a register variable for `d`:

```
boo() {
        register int a,b,c;

        first use of a
        statements that do not use d
        last use of a

        remaining statements, replacing all occurrences of d by a
        }
```

If instead the statements between the first and last use of `a` used the value of `d` (for instance, if the value of `d` were set there), you could still use this technique by inserting the statement:

```
a = d;
```

after the last use of `a` and before the first use of `d`.

Another, more readable way of doing this is to write the code as follows:

```
boo() {
        register int b,c;
        {
              register a;

              statements using a
        }
        {
              register d;

              statements using d
        }
}
```

In the above example, the compiler uses the `%ebx` register for both `a` and `d`.

Beware of using `goto` statements that jump into and out of scopes. If you use them, make sure you do not do something that can possibly lead to a computation on the wrong value.

Even if the scope of variables is not disjoint, it is sometimes beneficial to explicitly save and restore values in order to reuse a register variable. This makes the code somewhat more difficult to read, but can provide a much needed performance improvement.

Writing Linear Code

The Sun386i system executes linear code more efficiently than code that branches. This is because the 80386 has a 16-byte instruction pipeline, allowing it to fetch instructions in a look-ahead manner; the next instruction is ready to execute as soon as the previous one completes. This occurs only when the code is linear. When a branch is taken, the pipeline is flushed and the next instruction to execute is fetched from memory.

You can make very tight loops execute significantly faster by "unrolling" them into linear statements. Some compilers with global optimizers use this technique. The well known sieve of Eratosthenes benchmark program contains the following initialization loop:

```
for (K = &FLAG[0]; K <= LAST; K++) *K = TRUE;
```

Unrolled, the loop looks like this:

```
for (K = &FLAG[0]; K <= LAST; ) {
      *(K++) = TRUE;
      *(K++) = TRUE;
      *(K++) = TRUE;
      *(K++) = TRUE;
      *(K++) = TRUE;
}
```

This change provides a 9% improvement. The optimization is possible because the array being initialized contains a multiple of five entries (that is, 1,000,000). With some other number, the number of assignments placed in the loop might be different. Even if the number of entries is prime, you usually can unroll a loop using other methods.

Another thing to look for is the number of branches that are taken in a code fragment. In some cases you can change conditions or move expressions so that, in most cases, the final code branches less; this means fewer flushes of the pipeline. Sometimes you can change from this:

```
if (c) {
        linear statements
}
```

to this:

```
if (!c) {
        complementary linear statements
}
```

if you know that the condition, `c`, is usually false. This avoids the extra branch to get around the first set of statements. This is not an easy case to implement, but it can sometimes be useful.

Finally, you occasionally can move code so that linear sequences of statements are longer, thereby using the pipeline more efficiently. For example, you can change from this:

```
linear sequence 1
if (c1) statement;
linear sequence 2
if (c2) statement;
linear sequence 3
```

to this:

```
linear sequence 1
linear sequence 2
linear sequence 3
if (c1) statement;
if (c2) statement;
```

when the statements in *linear sequence 2* and *linear sequence 3* are not dependent upon what happens within the conditionals. This technique is especially effective when the size of the statements in the linear sequences is less than the full 16 bytes of the pipeline.

Replacing Complex Operations

Substituting complex operations with simpler ones increases code efficiency. For instance, replacing the expression `X*2` with `X+X` decreases program runtime, since addition is faster than multiplication. Such substitution is called *strength reduction.*

The Sun386i compiler does a good job of strength reduction for multiplication. It tries to use either shifts or special addressing modes to implement a multiplication by a constant rather than by using the relatively more expensive `imul` instruction. However, the Sun386i system does not yet perform strength reduction for division. For division operations having a divisor that is a power of two, you can substitute a shift right instruction. For example, you can replace:

```
        a = b / 16;
```

by:

```
        a = b >> 4;
```

This generates only three machine instructions instead of five and avoids the use of the `idiv` instruction.

Evaluating Conditions

The C language specifies that the logical operations `&&` and `||` are evaluated in a left-to-right order. This means that the left side of an expression is always completely evaluated before the right side is evaluated. In the case of `&&`, if the left side is false, the right side is never evaluated. In the case of `||`, if the left side is true, the right side is not evaluated.

In some cases, you might be able to determine that one condition in an expression will be true or false the majority of the time; you then can arrange the order of the tests to take advantage of this. For example, if you have a test to see whether a character is a space or a new line within a source program, you could use either of the following statements:

```
        if ((ch == ' ') || (ch == '\n')) ...
```

or:

```
        if ((ch == '\n') || (ch == ' ')) ...
```

The first case is more efficient because usually there are more blanks than new lines in a source program.

Caution: If the expression is more complicated, you must make sure that all variables are evaluated in the desired manner. For instance, if in the previous example the original code were:

```
        if (((ch = getchar()) == '\n') || (ch == ' ')) ...
```

you would have to change it to:

```
        if (((ch = getchar()) == ' ') || (ch == '\n')) ...
```

instead of:

```
        if ((ch == ' ') || ((ch = getchar()) == '\n')) ...
```

This commonly occurs when pre- or post-increment or decrement operations are used on the left side and the expression is rearranged.

Generating String Instructions

The 80386 has a set of instructions that are designed for string operations. These are generated in the compiler for structure moves. You can take advantage of these string operations for other operations, such as initializing an array.

For instance, if you have an array of 10,000 integers and want to set each to the value 17, you could use a loop that would cycle 10,000 times and index through the array. This could be extremely inefficient since it probably would not take advantage of the 16-byte pipeline. The less obvious and faster way is to make the compiler think it is moving a structure:

```
struct dummy { int x[9999] };

int a[10000];

boo() {
        a[0] = 17;
        *(struct dummy *)&a[1] =
           *(struct dummy *)&a[0];
        }
```

This completes the job in only six machine instructions.

Improving Loop Efficiency

Many programs spend a lot of time executing loops. To improve the efficiency of loop execution, make the loop termination condition a test for zero. Usually this means counting down to zero, rather than up to some other value. In other cases, more manipulation is required, but it is usually worth it.

For example, the loop:

```
for (i = 10; i; --i) {statements}
```

is more efficient (by one machine instruction) than:

```
for (i = 0; i != 10; ++i) {statements}
```

Some other generalities that can increase loop efficiency are:

- `while` loops are often more efficient than `for` loops.
- Using `int` variables for loop control sometimes is more efficient than using `short` or `char` variables.
- If you are using a register variable as an array index, making it an `int` rather than a `short` or `char` is more efficient.

The following example illustrates the greater efficiency of `while` loops compared to `for` loops:

```
for( i = 10; i; i--) ;
```

generates five machine instructions while:

```
        do {} while (--i);
```

generates only three instructions.

Using Assembler Code

If you've exhausted all other optimization methods and still must increase program speed, try using assembler code. You can submit assembler code to the C compiler by using the asm keyword. This is called *inlining*, and it works like this:

```
        asm( "assembler statement" );
```

Code asm keywords as you do function calls. You can put more than one assembler statement in the quotes, but you have to include \n (new-line character) in the string yourself. It is better to use one assembler statement per asm keyword.

When you use the asm feature, you can reference any global or external variable by name. The only way to reference local variables is to know their offset from %ebp. To get the offset, compile the program with the -S option and search the code generated for reference to the variable. The offset from %ebp depends on the position of the variable in the function and the number of register variables.

4.8. Communications Software

The standard SunOS system communications and networking facilities on the Sun386i system are listed below. For more information on these products, refer to *Network Programming on the Sun Workstation.*

- NFS (Network File System) — enables file sharing in a heterogeneous environment of machines, operating systems, and networks
- RPC (Remote Procedure Call) — provides a mechanism whereby a *client* (caller) process can have a *server* process execute a procedure call, as if the caller process had executed the call itself; the processes can be on the same or different machines.
- XDR (eXternal Data Representation) — a specification for portable data transmission that is part of the RPC mechanism, and is helpful when porting between different machines and processes
- rpcgen(1), which can generate both RPC and XDR code
- YP (Yellow Pages) — a distributed network look-up service that maintains a set of databases, fully replicated at several sites, that users can query

In addition, the Sun386i system comes with Ethernet TCP/IP network support.

4.9. Database Software

The Sun386i system also offers the following unbundled database software:

- SunUNIFY — relational database software that provides fourth-generation applications development facilities for interactive and networked applications
- SunSimplify — uses the SunView window environment to provide a variety of end-user tools that enable easier, interactive access to SunUNIFY

5

Porting Summary

5

Porting Summary

Chapters 3 and 4 contain a mix of information relevant to porting applications to the Sun386i system, and information of a general nature about the Sun386i system. This chapter is a synopsis of porting issues and tools. In addition, page 62 contains a porting checklist.

5.1. Summary of Porting Issues

Keep these things in mind when porting software to the Sun386i system:

UNIX-Based Versus Other Applications

You can easily port applications developed for Sun-2™, Sun-3, and Sun-4™ workstations to the Sun386i system because the architectures are source-code compatible. Applications written in C, FORTRAN, or Pascal on other UNIX systems are also source-code compatible (see System V Issues on the next page for a list of SunOS system calls that don't conform to the SVID). To port applications that do not run on UNIX-based systems, you must rewrite code.

Binary Incompatibility

The Sun386i system does not offer binary compatibility with System V or other 80386-based applications; with the exception of PC applications running in DOS Windows, you must recompile your programs on the Sun386i system.

Potential Byte Swap Problems

The order in which data is arranged in the Sun386i system's 32-bit registers is similar to some 32-bit processors (such as VAX systems) but is different from others (such as Sun-3 and IBM 360 systems). Byte swap problems can occur when:

- Reading binary data files created on one architecture on a different architecture
- Porting graphics between architectures
- Forgetting to include `+1` when calling `strlen()` and `malloc()`
- Passing messages over the network
- Performing character-to-integer or integer-to-integer conversions between architectures

COFF Use

The Sun386i system uses the UNIX System V Common Object File Format (COFF). The library `libld.a` contains functions to access and manipulate COFF object files.

as Assembler Differences

The Sun386i system uses the 80386 assembler `as(1)`. This assembler is different from 8086 assemblers with which you may be familiar; for instance, the assembler syntax for instructions such as `mov` places the source before the destination.

Archive Member Limitation

Archive members are limited to 14 characters on the Sun386i, but the maximum is 15 characters on Sun-3 systems. If you build a library that has a member with a 15-character name, the Sun386i system truncates the name. Truncation could produce duplicate member names.

Porting Large Programs

For very large programs with address spaces exceeding 2 gigabytes, addresses can appear to be negative because signed integers are represented in two's complement notation. Mixing pointers and integers carelessly could be dangerous.

System V Issues

The following SunOS system calls do not conform to the SVID:

- `creat`(2) and `open`(2V)
- `chown`(8)
- `utime`(3C)
- `kill`(1) and `kill`(2V)
- `mknod`(8)
- `fcntl`(2)

In addition, the following terminal interface specifications are not fully supported:

- 5-bit and 6-bit characters
- `VMIN` and `VTIME`
- Erase and kill characters

The System V and Berkeley Compatibility section on page 29 contains details about the above incompatibilities.

C Issues

Be aware of the following issues when porting C programs to the Sun386i system:

- The Sun386i C compiler, `cc`(1), aligns data on natural boundaries: `char` on byte boundaries, `short` on word boundaries, and `long` on doubleword boundaries. Problems with structure interpretation can result when porting between architectures.
- `cc` does not allow you to cast a structure to a scalar value.
- `cc` does not have a global optimizer and does not support the `-a`, `-align`, *float_option*, `-ffpa`, `-fsky`, `-vpa`, and `-J` options.
- The C preprocessor, `cpp`(1), predefines the symbols `i386`, `sun`, and `unix` for use in `#ifdef` statements; `cpp` does **not** predefine the statements `mc68000`, `m68k`, `mc68010`, `mc68020`, or `sun386`.

- The Sun386i system has a maximum shift count of 31. To avoid incorrect results, search through your code and check all shift operations. For each operation such as *value* << = *count*; where *count* could be greater than 31, insert the following code:

```
if (count > 31)
    value = 0;
else
    value <<= count;
```

XDR File Format

Use the XDR file format to enable your applications to run on other architectures within a Sun network.

SunView Facilities

When developing window applications for the Sun386i system, use SunView programming facilities.

Graphics Applications

Similarly, when developing or porting graphics applications that must access low-level graphics facilities, do not write directly to the frame buffer; instead, use Pixrect library routines. Consider the following when dealing with pixrects:

- Check code that draws manually into a pixrect; you might have to modify such code before it will port properly. The type of modification required depends on the particulars of the drawing operation.
- Perform manual operations (those not involving `libpixrect.a` routines) on a pixrect before converting the pixrect to 80386 format.
- `mem_create` creates an 80386-format pixrect on Sun386i systems.
- `mem_point` does not set the `MP_I386` flag. The pixrect is considered not flipped.
- To create an icon, use `mem_point` to make a pixrect connected to an existing static image or an image that you have created on the fly. You can use `DEFINE_ICON_FROM_IMAGE` to create static icons. All static icons are initially created in 680x0 format. The system converts static icons to 80386 format as soon as they are involved in a raster operation.

5.2. Summary of Porting Tools

There are two main porting tools:

- The `pr_flip` utility that corrects the byte-ordering problem that occurs when you execute 680x0 format graphics on a Sun386i system (see page 46 for details)
- The `lint`(1V) command, and in particular the `-c` option, which checks for casts that might not port correctly

In addition to these tools, you can also use:

- `adb`(1) — assembly language debugger
- `dbx`(1) — source-level debugger

- `dbxtool`(1) — window- and mouse-based source-level debugger
- `trace`(1) — for tracing system calls that a process makes

5.3. Checklist of Porting Procedures

If you perform the following procedures that apply to your situation, your applications should run successfully on the Sun386i system. If your application currently runs on a Sun system, this checklist assumes that you are porting it from a Sun-3 system, and that your program's access to the monitor screen is through the documented SunView and Pixrect interfaces. If the latter assumption in not true, see Frame Buffer and Graphics on page 20 and Window System and Graphics Support on page 45 for information helpful when dealing with graphics porting issues.

Sun386i Porting Checklist

DONE	*N/A*	
□	□	On the Sun-3 system, `lint`(1V) your source code and treat anything found as a potential problem.
□	□	If you are writing in assembly language and invoking C routines, note that the C compiler does not prepend an underscore character to external variable and function names.
□	□	Use the `grep`(1) command for `strlen` functions used in memory allocation routines and ensure that the null terminator is accounted for (see Byte Ordering on page 34).
□	□	Rewrite code that manipulates structure fields in memory and assumes 680x0 data alignment (see Data Alignment on page 38).
□	□	Rewrite assembly language routines and use the 80386 assembler definition. Also, use the `grep`(1) command on your C sources for the `asm` keyword and rewrite embedded assembly code. (Appendix B contains more information on the 80386 assembler definition and Appendix F contains information on register usage and stack frame format during function calls.)
□	□	Use the `byteorder`(3N) functions (for existing applications that do not use floating-point data) or the XDR mechanism (for new applications, or applications that do use floating point) to byte swap integer data in 680x0 files accessed by your program (see Message Passing Over the Network on page 35).
□	□	Make sure that application data flags and network protocols are architecture independent by using the XDR file format (or your own standard).

6

User Interface

6

User Interface

Windows, graphics, and the user interfaces to which they contribute are important features of all Sun systems. This chapter discusses windows and graphics on the Sun386i system and introduces some of its user interface features. It includes information about:

- The window system and window-based applications
- The on-screen help facility (both what comes with the Sun386i system and how you can add on-screen help to your own applications)
- Ease-of-use administration tools
- Using color in your applications and the `coloredit` tool

6.1. Window System

The following discussion divides the Sun window system into three areas:

- The *window substrate*, comprised of low-level facilities for user interface programming, such as the Pixrect library
- The *window toolkit*, containing most of the programming facilities (tools) you will need; SunView occupies this level
- *Window-based applications*, consisting of standard system applications included in SunView (such as Mail, Text Editor, and Commands)

The following sections describe each of these areas as it pertains to the Sun386i system.

Window Substrate

The window substrate in Sun systems is provided by the Pixrect library of raster operation (RasterOp) routines, `libpixrect.a`. These routines work the same way on the Sun386i system as on other Sun systems. If you are porting programs that currently run on other Sun machines to the Sun386i system, you do not have to do anything special to make them behave properly (byte swapping is handled for you by system software—refer to the *Pixrect Reference Manual* for details). If you are porting programs that do not currently run on Sun systems, the *Pixrect Reference Manual* and the *SunView Programmer's Manual* should provide you with the information you need.

Window Toolkit

The window toolkit for all Sun systems is SunView. If you are already familiar with SunView, you will notice that SunView 1.75, the version shipped with the Sun386i system, has some changes and new features. First, it has benefited from a number of minor changes, for example, some aesthetic improvements in the appearance of window icons. The `Put`, `Get`, and `Delete` commands on the standard text subwindow menu have been renamed to `Copy`, `Paste`, and `Cut`, respectively. (Note in Figure 3-2 on page 22 that the legends on the tops of the [L6], [L8], and [L10] keys of the Sun386i system keyboard are also [Copy], [Paste], and [Cut].)

Among new features in SunView 1.75 are enhancements to the color facilities. In particular, end users can set colors for such things as background and foreground with the `coloredit`(1) utility, described on page 104. Moreover, you can now set background and foreground colors for your application's panel items through dynamic manipulation of the panel's colormap.

A new facility provided in SunView 1.75 is the alert package. Your applications can use this package to display "alert boxes," panels with buttons and fill-ins letting users select an option to proceed or take some other action appropriate to the situation.

The Sun386i system also uses alert boxes for its on-screen help facility. The description of this facility, which has a programmatic interface, starts on page 76.

Window-Based Applications

In addition to the standard SunView programs—`defaultsedit`(1), `mailtool`(1), `textedit`(1), `cmdtool`(1), `console`(4S)—the Sun386i system also contains:

- `organizer`(1), an interactive, graphical interface to the SunOS file system, described on page 68
- On-screen help in the form of:
  - Revised kernel error messages — about 40 messages reworded for clarity
  - Spot Help — cursor-sensitive help that describes the object currently under the mouse pointer
  - Help Viewer — a more detailed description of tasks associated with specific window objects (the program name is `help_viewer`)

  Section 6.2, starting on page 73, describes these facilities and their interfaces in greater detail.
- `snap`(1), a user interface to a new set of system administration facilities intended to be easier to use than traditional SunOS facilities; page 98 provides more information about `snap`.
- `coloredit`(1), a utility that lets users set and change foreground and background colors for applications and icons (described on page 104)
- `dos`(1), a program that provides windows for the running of PC applications; Chapter 7 briefly describes the `dos` user interface.

Figure 6-1 on the following page shows the relationship between the various pieces of window system and graphics software on the Sun386i system.

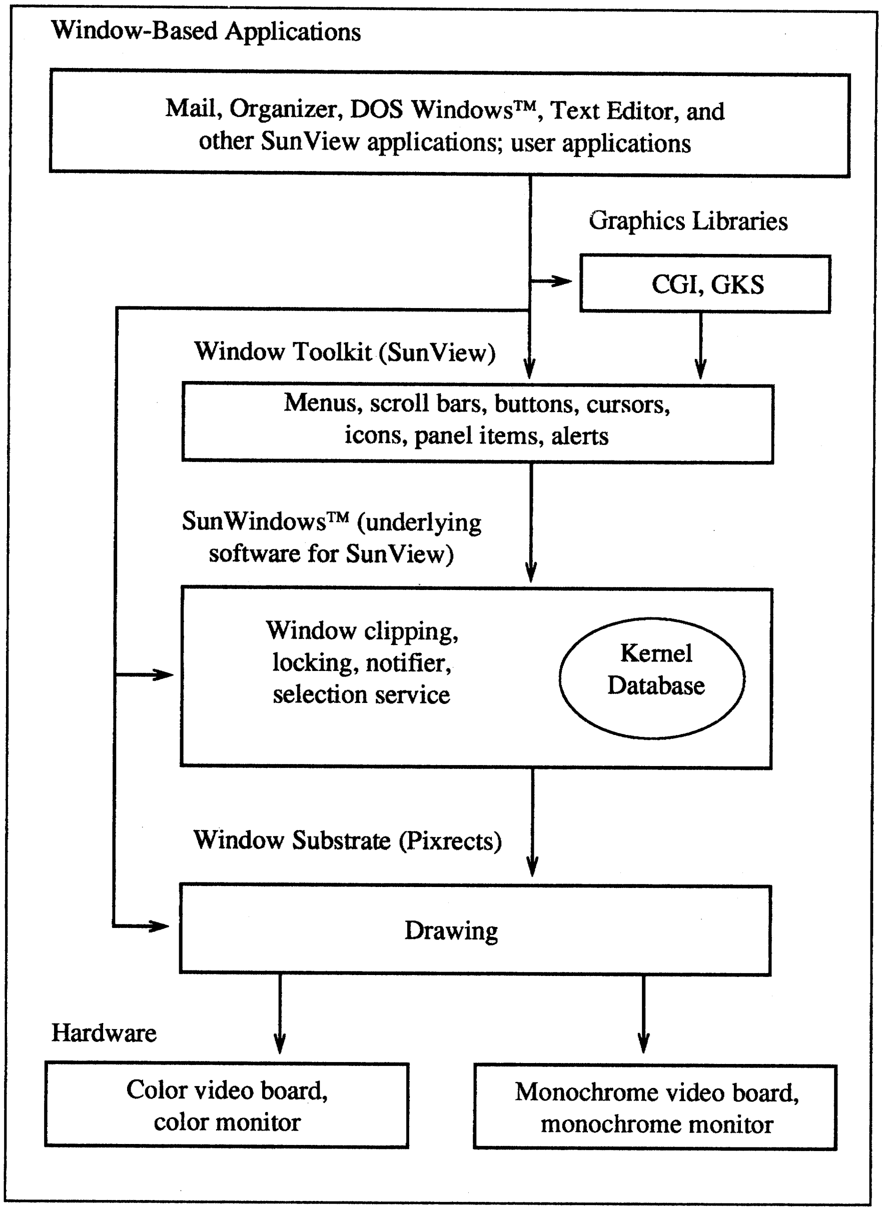

Window-Based Applications
Mail, Organizer, DOS Windows™, Text Editor, and other SunView applications; user applications
Graphics Libraries
CGI, GKS
Window Toolkit (SunView)
Menus, scroll bars, buttons, cursors, icons, panel items, alerts
SunWindows™ (underlying software for SunView)
Window clipping, locking, notifier, selection service
Kernel Database
Window Substrate (Pixrects)
Drawing
Hardware
Color video board, color monitor
Monochrome video board, monochrome monitor

Figure 6-1 *Window System and Graphics Software*

As shown above, the Sun386i system supports a version of the ANSI Computer Graphics Interface (CGI) standard graphics package for generation of two-dimensional images, called SunCGI. In addition, the Sun386i system supports SunGKS, the well-established Graphical Kernel System (GKS) standard for interactive two-dimensional graphics. Both SunCGI and SunGKS are unbundled products.

The next section discusses the Sun Organizer™, one of the window-based applications shown above, which you can use to create icons for your application's files.

The **organizer** Program

Organizer (organizer(1)) displays the contents of directories by using icons to denote file types. Pop-up menus, panel buttons, and property sheets provide SunOS file system commands such as mv(1), cp(1), rm(1), lpr(1), edit(1), open(2V), rename(2), chmod(1V), find(1), and mkdir(1).

Users can display directories with or without icons, and they can sort the contents of these directories by name, file type, size, or date. In addition, with the Show Map feature users can graphically view or browse the file system hierarchy in one window. You can create icons for your application's files and include them in the .orgrc file, described in the following section. When you do, organizer automatically displays your file-type icons in its Show List and Show Map windows.

For background information on the organizer interface, refer to the *Sun386i User's Guide*, *Sun386i Advanced Skills*, and the on-screen *Organizer Handbook*.

Displaying File Types for Your Applications

organizer(1) can display three-color icons representing your application's files if you:

- Create icons for your file types.
- As part of the installation procedure, put your .orgrc file in a subdirectory beneath your application directory in *application_name*/share/data and your icon files in *application_name*/share/icons . If there's room, *application_name* should be created in /usr/local. (On Sun386i systems, /usr/local is a symbolic link to /files/local. Pages 144–145 provide more information about distributing software for the Sun386i system, and Appendix C describes the Sun386i file system layout.)
- As part of your installation notes, instruct system administrators to create a volume for your application (page 197 describes volumes and page 99 lists steps to create them) and tell users to issue the cat(1) command to append your .orgrc file to their individual version of the file in /files/home/*groupname*/*username* after your application is installed. Then tell users to quit from and re-enter Organizer to see your application's icons. (source(1) does not work on .orgrc.)

Users automatically get a copy of .orgrc when a new user account is created. After they append your .orgrc to their own copy of the file, users can open a file or run an application by double-clicking on the icons you supply. The default version of .orgrc is in /files/home/users/users/defaults. Then, for each user account created, the system puts a copy of .orgrc in the directory /files/home/*groupname*/*username*.

The steps to create and incorporate a file-type icon into organizer are:

1. Determine the name-matching expression to identify your file type. For example, the defaults that come with the system are *.c for C program files, *.h for header files, *.o for object files, and *.icon for icon files. It is the responsibility of the system administrator to rename duplicate file-type extensions that might occur.
2. Create one set of icons per file type that you are defining with the iconedit(1) utility. To enable three-color icons, you must create:
   - A background icon for the icon's background and
   - A foreground icon, which goes on top of the background icon

   To create two-color icons, you need only specify a background or foreground icon.

3. Quit from the organizer if it is running.
4. Add entries describing your icons to the version of .orgrc in your home directory (/files/home/*groupname*/*username*). The format of .orgrc is shown in the following sections.
5. Re-enter organizer to view the icons created or changed.

After restarting organizer, you can use coloredit(1) to change the colors of icons. When you quit from organizer, the changes made are saved to the version of .orgrc in your home directory.

Once you determine the colors you like best, make a copy of your home .orgrc file for distribution. During installation, move this copy to /usr/local/*application_name*/share/data/.orgrc if there's room (see pages 144-145 for details).

.orgrc Parameters

.orgrc files contain two kinds of parameters:

- File-type parameters — You must create one set of file-type parameters for each icon accompanying your software. These parameters are described in the next section, with required parameters shown in bold.
- Color-palette parameters — These parameters define the default colors that the Sun386i system uses for directory, text, executable, and device files. You can use the colors supplied by these parameters for your icons, add to this file if your application uses files other than the four default types, or ignore this section of .orgrc and use your own RGB values in your file-type parameters. These parameters are described on the next page.

All .orgrc parameters, as well as all values for parameters, are case insensitive.

File-Type Parameters

Begin File Type Definition

These four words are required to denote the start of each file type within .orgrc.

Name

The expression used for name matching; you can use any valid SunOS wildcard. For instance, for a file type ending with .pbj, you could enter *.pbj as a value for this parameter. Another example is *,s for SCCS (Source Code Control System) files. Alternatively, you could enter the exact file name, as with the core file type provided with the Sun386i system.

Background Icon

The path name of the background icon that organizer will display.

Foreground Icon

The path name of the foreground icon (placed on top of the background icon) that organizer will display.

Name Color

The RGB values for the color of the text of the file name as it appears on the icon. Also used to color the rectangle that surrounds a file when it is selected.

`Icon Background Color`
The RGB values for the background color of the icon.

`Icon Foreground Color`
The RGB values for the foreground color of the icon.

`Highlight Name Color`
The RGB values for the color of the text of the file name when the file or directory is selected (highlighted).

`Execute Application`
The name of the application to call to open or execute this file type. Some file types that users should not open are object files, intermediate database format files, libraries, and binary data files.

If the application that you're opening accepts file name arguments, you can use the `$(FILE)` keyword to tell `organizer` to open that application's files when users double-click on a file name. Just use `$(FILE)` instead of the file name in the command line, as shown in the following example:

```
    Execute Application = textedit $(FILE)
```

`Edit Application`
The name of the application to call to edit this file type. If you do not supply a value for this parameter, users will be unable to edit any files of this file type from within `organizer`. As with `Execute Application`, you can specify the `$(FILE)` keyword so that users can edit the selected file.

`Print Application`
The name of the application to call to print this file type. If you do not supply a value for this parameter, users will be unable to print any files of this file type from within `organizer`. If the application can print files automatically, you can include the `$(FILE)` keyword to enable users to automatically print the application's files through `organizer`.

The `Name Color`, `Icon Foreground Color`, `Icon Background Color`, and `Highlight Name Color` parameters are optional because you can use `coloredit` to determine icon colors; when you quit from `organizer`, it automatically writes the colors chosen to `.orgrc` in your home directory. You can then include the RGB values from your home `.orgrc` in the version that you supply with your application.

`End File Type Definition`
These four words are required to denote the end of the definition.

Color Palette Parameters

Instead of providing numerical values for your icon colors in the preceding color-related parameters, you could use the names of the color palette parameters shown below for directory, text, executable, and device file icons. You might want to use these parameter names instead of RGB values in your file-type definitions because:

- It's easier to use them than to use trial and error to determine colors for your applications.
- The colors of your directory, text, executable, and device icons will match the default values for SunView versions of these file types; the graphic of your icons rather than their colors will differentiate them.
- If users change the color palette RGB values, your icons will match the colors they choose.
- You might not want to use any more of the colormap on your icons, particularly if your application uses a lot of color (the Sun386i system permits a maximum of 256 colors at one time).

The default color palette portion of `.orgrc` supplied with the Sun386i system follows.

```
Begin Color Palette
 Background Color = 255, 255, 255
 Directory Name Color = 0, 146, 236
 Directory Icon Foreground Color = 255, 227, 185
 Directory Icon Background Color = 114, 45, 0
 Directory Highlight Name Color = 255, 247, 9
 Text Name Color = 0, 166, 143
 Text Icon Foreground Color = 255, 255, 255
 Text Icon Background Color = 0, 0, 0
 Text Highlight Name Color = 255, 255, 0
 Executable Name Color = 255, 0, 104
 Executable Icon Foreground Color = 243, 255, 254
 Executable Icon Background Color = 157, 162, 187
 Executable Highlight Name Color = 255, 247, 9
 Device Name Color = 111, 111, 111
 Device Icon Foreground Color = 243, 255, 254
 Device Icon Background Color = 157, 162, 187
 Device Highlight Name Color = 255, 255, 0
 Scrollbar Color = 0, 87, 185
End Color Palette
```

Most of the color palette parameters correspond to file-type parameters. For instance, to use the default colors for a text file icon, include the lines below in your file-type definition:

```
Name Color = Text Name Color
Icon Foreground Color = Text Icon Foreground
  Color
Icon Background Color = Text Icon Background
  Color
Highlight Name Color = Text Highlight Name
  Color
```

The first and last color palette parameters, Background Color and Scrollbar Color, do not have file-type counterparts.

If your application uses files that don't fit one of the four default categories, you can add to the color palette section. However, do not change the defaults that are there, since these are the colors that Sun wants users to see; users can modify these colors if they so choose.

Alternatively, you might want to provide your own icon colors to make them stand out (but be aware that users can change these also). If so, provide RGB values instead of color palette parameter names in your file-type definitions.

Sample .orgrc Entries

This section provides sample entries for the .orgrc file.

```
Begin File Type Definition
 Name = *.c
 Background Icon = /vol/application_name/
   share/icons/bkgd.icon
 Foreground Icon = /vol/application_name/
   share/icons/frgd.icon
 Name Color = 236, 121, 85
 Icon Background Color = 205, 205, 254
 Icon Foreground Color = 51, 19, 92
 Highlight Name Color = 255, 247, 9
 Execute Application = cmdtool vi $(FILE)
 Edit Application = cmdtool vi $(FILE)
 Print Application = lpr $(FILE)
End File Type Definition

Begin File Type Definition
 Name = *.icon
 Background Icon = /vol/application_name/
   share/icons/bkgd.icon
 Foreground Icon = /vol/application_name/
   share/icons/frgd.icon
 Name Color = 124, 61, 140
 Icon Background Color = 243, 255, 254
 Icon Foreground Color = 157, 162, 187
 Highlight Name Color = 255, 254, 8
 Execute Application = iconedit $(FILE)
 Edit Application = iconedit $(FILE)
End File Type Definition

Begin File Type Definition
 Name = *.bits
 Background Icon = /vol/application_name/
   share/icons/bkgd.icon
```

```
  Foreground Icon = /vol/application_name/
    share/icons/frgd.icon
  Name Color = 124, 61, 140
  Icon Background Color = 243, 255, 254
  Icon Foreground Color = 157, 162, 187
  Highlight Name Color = 255, 254, 8
  Execute Application = fontedit $(FILE)
  Edit Application = fontedit $(FILE)
End File Type Definition
```

6.2. On-Screen Help Facilities

In addition to man pages, the Sun386i system offers three areas of on-screen messages:

- Detailed instructions on how to create a new user account and how to log in
- Improved kernel error messages
- Cursor-sensitive help for SunView applications

The log-in screens are part of the new administration features, and are described more fully on page 99. This section describes kernel error messages and cursor-sensitive help for applications, including how to add to or change both types of these messages.

Kernel Error Messages

About 40 of the most frequently displayed kernel error messages have been reworded for the Sun386i system. The original messages are intercepted by a message daemon and used as keys into a message text file. The keyed messages, rewritten for clarity, then are delivered to a log file, to a SunView window, or to both.

Substituting Error Messages

There are several possible reasons for changing error message output:

- To translate the messages that Sun has reworded into another language (refer to the Native-Language Messages section on page 154 for more information)
- To intercept and reword the output of additional error messages generated by the kernel (those sent to `/dev/klog` by the `syslogd`(8) daemon)
- To intercept and reword the output of error messages generated by local programs, provided the program sends its messages to `/dev/log` via `syslog`(3)
- To intercept and reword the output of messages placed in the internet socket by programs on other systems

The steps below apply to the addition of substitute messages (the latter three cases).

1. Look through kernel source code (if you have the appropriate license), or the source code for the program containing messages that you want to replace. You can only substitute messages for programs that send their messages to `/dev/log` via `syslog`.

2. Add the text of the existing, unexpanded messages that you want to replace to /etc/In, preceding each message with a unique number. (You also can include suppression instructions for these messages, as described later in this section.)
3. Add the substitute text for these messages to /etc/Out, preceding each message with the number corresponding to the original message in /etc/In.
4. Use the kill -1 command on the syslogd process to activate the changes made.

You don't have to use the default files /etc/In and /etc/Out for original and substitute text, respectively, but it's easier if you do. If you use other files, you must include the file names in /etc/syslog.conf. This file tells the system log daemon, syslogd(8), the information needed to perform the translation or suppression. syslogd intercepts error messages, checks /etc/In and the file you specify in /etc/syslog.conf (if any) for the text of those messages, and either suppresses, substitutes, or displays the original messages.

If you use files other than /etc/In and /etc/Out, you must add an entry to /etc/syslog.conf that contains the following five fields, separated by tabs:

```
translate source facility input_file output_file
```

translate — Denotes a translation entry.

source — Specifies the source(s) of an error message, separated by commas; recognized sources are:

- klog, indicating a kernel message sent to /dev/klog
- log, indicating a message generated by a local program and sent to /dev/log
- syslog, indicating a message placed in the internet socket by programs on other systems
- *, indicating all three of the above sources

facility — Specifies messages generated by other system facilities, separated by commas:

- user, indicating messages from user processes. This is the default for messages from programs or facilities not listed in syslog.conf.
- kern, indicating messages from the kernel
- mail, indicating messages from the mail system
- daemon, indicating messages from system daemons such as ftpd(8) and routed(8)
- auth, indicating messages from the authorization system (login(1), getty(8), and su(1))
- lpr, indicating messages from the line printer spooling system
- cron, indicating messages from the cron/at facility
- mark, indicating messages produced internally by syslogd
- *, indicating all facilities except mark

input_file — Specifies the name of the file that maps numbers to `printf`-format error messages, and indicates whether or not a message will be suppressed. Unless you are completely suppressing a message, the numbers you use in this file must have an identically numbered counterpart in *output_file* for translation or suppression to occur. Also, you must include the exact, unexpanded text of the original message. *input_file* must be in `xopen`(5) format.

You can indicate one of several levels of suppression for an error by using the following specifications in *input_file:*

- `(NONE)`, indicating complete message suppression
- (*n*), indicating this message will be displayed no more than once every *n* seconds
- `( )`, indicating no suppression

A sample *input_file* is shown in the Sample Input and Output Files section, which follows.

output_file — Specifies the name of the file that provides numbered, `printf`-format messages that will replace the messages listed in *input_file.* As with *input_file, output_file* must be in `xopen`(5) format. You can change the order of the variables displayed by replacing the initial `%` of a format phrase with a `%`*num*`$`, where *num* is a digit specifying which *input_file* variable (for example, 1, 2, 3 for first, second, and third variable) to insert at a given place in *output_file.* A sample *output_file* is shown in the next section.

Sample Input and Output Files

This section shows a sample input file, corresponding output file, and entry in `/etc/syslog.conf` to define these files to `syslogd`.

Sample Input File

The following entries are part of `/etc/In`.

```
$quote "
9   "(NONE)\n%s: write failed, file system is
full\n"
10  "(10)NFS server %s not responding still try-
ing\n"
11  "(100)NFS %s failed for server %s:L %s\n"
```

The `(NONE)` in message 9 ensures that this message will never be displayed or sent to a log file. (Therefore, `/etc/Out` does not require a corresponding message.) The `(10)` in message 10 and the `(100)` in message 11 tell `syslogd` to display or send these messages no more than once every 10 and 100 seconds, respectively.

Sample Output File

The following entries are part of `/etc/Out`.

```
$quote "
10  "Network file server %s not ok. Check your
cable\n"
11"Network file server %2$s down (%1$s, %3$s)\n"
```

Message 10 will be displayed as shown on the previous page, with a server name replacing the variable. Message 11 will be translated and the file server and error type information reordered. For instance, if the original message is:

```
NFS getattr failed for server local_host: RPC:
Timed out
```

The translated message will be:

```
Network file server local_host down (getattr,
RPC: timed out)
```

because `getattr` is the first variable (specified by `%1$s` in the output file), `local_host` is the second variable (specified by `%2$s`), and `RPC` is the third variable (specified by `%3$s`) in the original message.

Sample Translation Entry

If you just add to `/etc/In` and `/etc/Out` instead of creating your own input and output files, then you don't have to add any entries to `/etc/syslog.conf`. If, however, you do create your own input and output files, you'll have to include an entry similar to the one below to `/etc/syslog.conf`.

`translate * *` *input_file output_file*

Remember, these fields are separated by tabs. The two asterisks denote all valid sources and all valid facilities; you might want to routinely use two asterisks as a short cut, instead of specifying sources and facilities individually.

Spot Help and Help Viewer Overview

The Sun386i Help facility consists of two parts: Spot Help and Help Viewer.

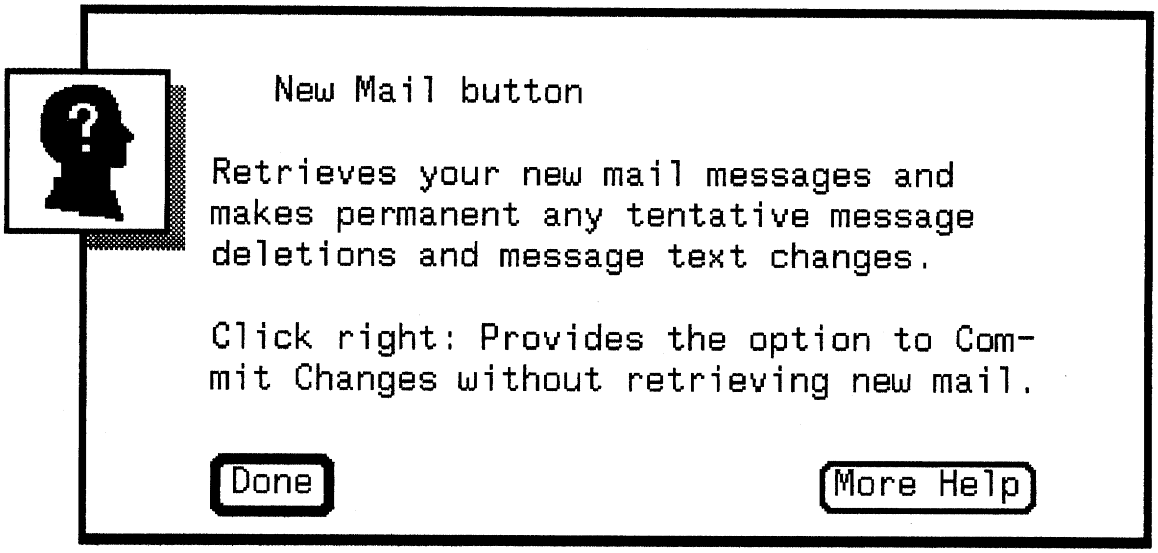
Spot Help displays quick explanations of SunView application objects. These objects can include windows, icons, menu items, scroll bars, and panel items. Spot Help is cursor-sensitive. To invoke Spot Help on a panel button, for example, a user points to the button with the mouse and then presses the [Help] key (Figure 3-2 on page 22 shows the Sun386i keyboard). A brief message explaining the button's function then appears in a pop-up window (also called an alert box).

Help Viewer (the program name is `help_viewer`) is a SunView application that displays more detailed information in the form of *handbooks*. While Spot Help might explain a particular button, a Help Viewer handbook might contain an introduction to the relevant application, as well as step-by-step instructions for accomplishing specific tasks using that application.

Spot Help and Help Viewer work together. If a Spot Help window does not provide enough information to answer a question, additional information is often available by selecting the More Help button in the bottom right corner of many Spot Help windows. Help Viewer then displays a page of the applicable handbook.

You can provide both Spot Help and Help Viewer handbooks for your application. The sections Spot Help Interface and Help Viewer Interface later in this chapter provide more information.

Figure 6-2 below shows a sample Spot Help window for the `mailtool` facility.



New Mail button

Retrieves your new mail messages and makes permanent any tentative message deletions and message text changes.

Click right: Provides the option to Commit Changes without retrieving new mail.

Figure 6-2 *Sample Spot Help Pop-Up for the* New Mail *Button*

The above pop-up appears when the mouse pointer is on the New Mail button and the [Help] key is pressed. The top line identifies the object for which help was requested, and the message text below describes the object's function and how to use it. The buttons at the bottom provide users with one or two exit options:

- Click left on the Done button. This dismisses the Spot Help window.
- Click left on the More Help button. This opens a Help Viewer window and displays a page in a Help Viewer handbook, one containing additional, more basic information about how the Help object is used. Figure 6-3 on the next page shows a sample Help Viewer page, the one displayed when the More Help button shown in Figure 6-2 is pressed.

Most Spot Help windows contain the More Help button. The standard set of handbooks that users can access by selecting More Help are listed below.

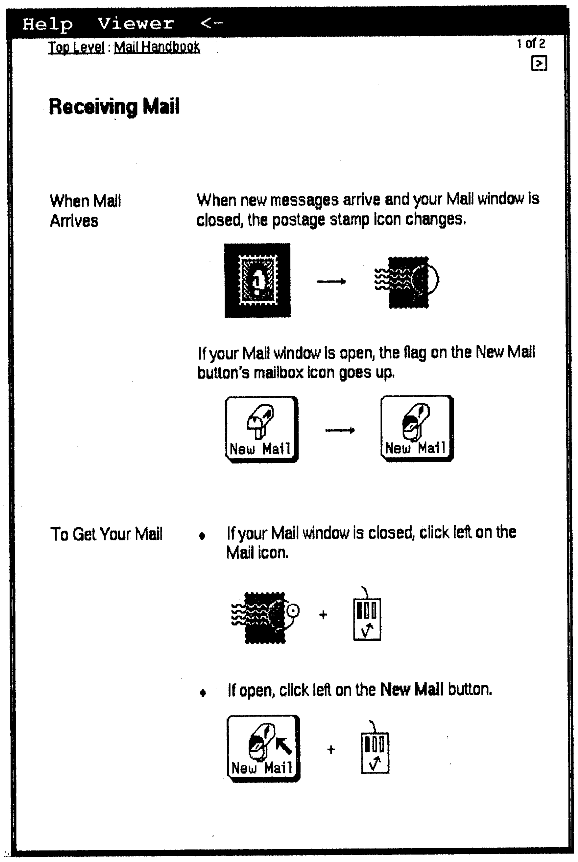
Organizer Handbook (`organizer`)
Mail Handbook (`mailtool`)
Help Handbook (Spot Help and Help Viewer documentation)
DOS Windows Handbook (`dos`)
SNAP Handbook (`snap`)
Text Editor Handbook (`textedit`)
Color Editor Handbook (`coloredit`)
Desktop Handbook (SunView basics)

Each of the handbooks on the Sun386i system follows a common organizational scheme:

1. A handbook contents page, listing each of the handbook's topics and its index

2. An introduction to the application, including a labeled diagram of its user interface and basic conceptual information; typically 5 to 10 pages long
3. A series of "how-to" topics designed to help users quickly accomplish new tasks using the application; handbooks typically contain four to eight topics of two to four pages, each illustrating several closely related procedures
4. An index of 100 to 200 entries for concepts, objects, and procedures discussed in the handbook

Help Viewer handbooks support high-quality text and graphics, as well as hypertext cross-referencing capabilities (described in the next section). Figure 6-3 shows the first page of a two-page document on Receiving Mail, in which the **New Mail** button is discussed and illustrated in the context of a procedure, getting your mail. This graphical and procedural approach to presenting basic information about SunView applications characterizes all of the Help Viewer handbooks.



Top Level : Mail Handbook

Receiving Mail

When Mail Arrives — When new messages arrive and your Mail window is closed, the postage stamp icon changes.

If your Mail window is open, the flag on the New Mail button's mailbox icon goes up.

To Get Your Mail

- If your Mail window is closed, click left on the Mail icon.
- If open, click left on the **New Mail** button.

Figure 6-3 *Sample Help Viewer Handbook Page*

Following Hypertext Links

Notice that the Top Level and Mail Handbook items in the page header are underlined. Underlined items indicate the presence of *hypertext links*. A hypertext link is a connector that leads from a particular place in one document to a particular place in another. By going through a menu or double clicking (left) directly on such links, the referenced item appears in the viewer.

Following the Mail Handbook link, for example, displays the handbook's contents page, listing all the topics. Each listed topic is linked to the corresponding document so that a user can display any topic in the handbook directly from the contents page.

Following the Top Level link calls up the Top Level table of contents. This lists all of the handbooks available in Help Viewer. Each handbook listed is linked to the corresponding contents page so that, again, users can follow the links and go directly to the handbook they want. A Master Index of all procedures and objects described in the standard Sun386i handbooks is also accessible from the Top Level.

Finally, note the right arrow button below the page number in the upper-right corner. Page buttons appear on every page and are graphical links for turning pages. These buttons function with just one click.

Supplying Help for Your Applications

The sections Spot Help Interface (starting on the next page) and Help Viewer Interface (starting on page 86) describe how to create help files for your application. Much of this information, particularly the Help Viewer Interface, is also available on line in the *Help Writer's Handbook.* Unlike the other handbooks supplied with the Sun386i system, you do not automatically receive the *Help Writer's Handbook*; it is part of the optional Developer's Toolkit software (described on pages 11-12). If you have the Developer's Toolkit, you can access the handbook by performing the following steps:

1. Enter **`cluster`** to see if the `help_guide` cluster is on your system. If it is, go to step 3.
2. If `help_guide` is not on your system, enter **`loadc help_guide`**, respond to the prompts displayed, and insert the tape or diskette specified. If there is enough disk space, `loadc` adds the cluster to your system in `/files/loaded/devel` directory. (If there is not enough disk space, you can remove clusters with the `unload`(1) or `unloadc`(1) commands, described on pages 12-13).
3. Edit the `Top_Level` file in the default help directory, initially `/vol/help.master/language/USA-English`, to include the line:

   ```
   'Help_Writer\'s Handbook' [help_guide/
   Help_Writer's_Handbook 1]
   ```

   and save the file. You must precede the apostrophe in `Writer's` by a backslash so that it won't be mistaken for the single quote delimiting the end of the title. The Adding Handbooks to the Top Level section on page 95 describes `Top_Level` in greater detail.
4. Follow the Top Level link on any Help Viewer page, and then follow the Help Writer's Handbook link.

Directory Structure for Help Files

With the exception of the *Help Writer's Handbook* described in the preceding section, all help files provided with the Sun386i system are in subdirectories of `/usr/lib/help`:

- Spot Help files are in `/usr/lib/help/language/USA-English`
- Standard handbook files are in `/usr/lib/help/language/USA-English/`*handbook*

- Template subdirectories and initialization files for creating handbooks and an extra copy of the Top Level file for linking handbooks into the Help system are in `/usr/lib/help/format/`*format_name*`/templates`

Subsequent sections give more information about individual files in each of these directories.

To enable easier, more efficient network-wide use of help files, the system automatically makes help files accessible to all users through the `/vol/help` directory. `/vol/help` contains links to the physical location of help files, and is the default directory where the Help system looks for Spot Help and handbook files. The rest of this description uses the `/vol` path name when referring to the location of help files. Pages 197 and 205 provide more information about volumes.

The Help system is designed to work regardless of the location of help files, provided that:

1. A default help directory contains the Spot Help, handbook, and Top Level and initialization files (or links to these files).
2. The Help system knows the location of the default help directory.

You can use Defaults on the SunView Menu to notify the Help system of a change in the default help directory. You must then copy the links in the directories below `/vol/help` to the directory chosen. From `/vol/help`, you can use the command `cp -r` *new_default_help_directory* to recursively copy all of the links you'll need to the default directory specified.

It might be easier to use a different default directory while you are developing your help system. However, after customers install the help provided with your application, it is preferable that users access it through `/vol/help`. Pages 96–98 provide specifics.

Spot Help Interface

This section explains how to create Spot Help messages for text subwindow, panel, canvas, alert, tty, and menu window objects, as well as for individual menu, scroll bar, and panel items. It assumes you are familiar with SunView programming concepts; for more information, consult the *SunView Programmer's Guide*.

The two basic steps to include Spot Help for a window object follow. (The first step is for an applications developer and the second is for a writer):

1. Add the `HELP_DATA` attribute to the object or to an item within the object. You can add this attribute like other SunView attributes, such as through a null-terminated attribute list.
2. Write the help file in the format specified in the `.info` File Format section on page 84.

When a user presses the [Help] key, the `HELP_DATA` attribute is retrieved from the current window or item. The text specified by the `HELP_DATA` value is then displayed in the Spot Help window. The following sections describe each step in greater detail.

HELP_DATA Attribute

The value for the HELP_DATA attribute must be a two-part string, enclosed in quotation marks, in the format

"*file:keyword*"

file is the name of the text file containing the help description. *file* must be located in the default help directory (see page 79) and must end with the suffix .info (such as mailtool.info). Although all Spot Help files must end with the .info extension, include only the base of the file name, **not** the extension, as the value of the HELP_DATA attribute. The Help mechanism automatically appends the .info extension to the file name that you supply, and then looks in the default help directory (/vol/help initially) for that file.

keyword is a word within the .info file that is associated with the specific help text that will appear when help is requested. Each .info file can contain multiple keywords, but no two keywords can be alike within the same .info file.

For example, a HELP_DATA attribute could be:

```
HELP_DATA, "accounting:w4"
```

When help is requested on this object, the Help facility:

1. Finds the accounting.info file.
2. Locates the w4 keyword.
3. Displays the text associated with that keyword in a Spot Help window.

The .info File Format section on page 84 contains more details about the structure and placement of .info file text. The next section describes how you can use the HELP_DATA attribute to make your Spot Help messages more helpful for users.

Providing More Specific Spot Help

You can change the HELP_DATA attribute of various window objects to suit particular circumstances, for instance if a menu item is active or disabled, or a frame is open or iconic. If you do, you can provide users with more context-sensitive Spot Help, as shown in this section.

HELP_DATA for Active and Disabled Objects

For example, you might give all disabled objects (such as greyed-out menu items) a new HELP_DATA attribute where you disable them in the code, and again where you activate them, as shown below:

```
/* this menu item invokes a save function */
Menu_item      mi_save;
          .
          .
          .
/* here the save function becomes disabled */
menu_set (mi_save, MENU_INACTIVE, TRUE,
          HELP_DATA, "progname:mi_save_inactive",
          0);
          .
          .
          .
/* and here it becomes active */
menu_set (mi_save, MENU_INACTIVE, FALSE,
             HELP_DATA, "progname:mi_save",
             0);
```

A correlating Spot Help message for the "save" function above could resemble the following:

```
Save menu item
Stores the current version of the file you have
loaded.
```

Spot Help for when the save function is disabled could be:

```
Save menu item (disabled)
Stores the current version of the file you have
loaded. This item is currently disabled because
you have not loaded a file.
```

Multiple entries can share a common Spot Help message, due to the flexibility of the .info file format (delineated on pages 84–85). For instance, a .info file for the previous example looks like:

```
:mi_save
Save menu item
Stores the current version of the file you have
loaded.

:mi_save_disabled
Save menu item (disabled)
Stores the current version of the file you have
loaded. This item is currently disabled because
you have not loaded a file.
```

Alternatively, a single-message version, for which the two keywords share the same Spot Help message, might work just as well:

```
:mi_save mi_save_disabled
Save menu item
Stores the current version of the file you have
loaded. This item is disabled if you have not
loaded a file.
```

HELP_DATA for Open Frames and Icons

You also could include HELP_DATA attributes for frames that are open and those that are icons (closed). The following sample program creates a base frame and then interposes an event function in front of the frame's normal event handler. This makes the program aware of when the frame opens or closes, as well as when the program should change the frame's HELP_DATA attribute.

```
#include <suntool/sunview.h>
#include <suntool/help.h>
#include <stdio.h>

main(argc, argv)
    int             argc;
    char            **argv;
{
    Frame           frame;
    Notify_value    sample_interpose();

   /* create frame using command-line arguments */
   frame = window_create(0, FRAME, FRAME_ARGS,
     argc, argv, 0);

   /* set HELP_DATA depending on whether frame is
      open or iconic */
   if ((int)window_get(frame, FRAME_CLOSED)) {
      window_set(frame, HELP_DATA,
        "progname:frame_iconic", 0);
   } else {
      window_set(frame, HELP_DATA,
        "progname:frame", 0);
   }

   /* interpose in order to spot future open/close
      events */
   (void)notify_interpose_event_func(frame,
     sample_interpose, NOTIFY_SAFE);

   window_main_loop(frame);
}

static Notify_value
sample_interpose(frame, event, arg, type)
    Frame           frame;
    Event           *event;
    Notify_arg      arg;
    Notify_event_type type;
{
    int             initial_state, current_state;
    Notify_value    value;

   /* get frame's state */
   initial_state = (int)window_get(frame,
     FRAME_CLOSED);

   /* handle the event */
   value = notify_next_event_func(frame, event,
     arg, type);
   /* if the frame's state has changed, change the
      HELP_DATA */
   current_state = (int)window_get(frame,
     FRAME_CLOSED);
   if (initial_state != current_state) {
      if (current_state) {
         window_set(frame, HELP_DATA,
           "progname:frame_iconic", 0);
      } else {
         window_set(frame, HELP_DATA,
           "progname:frame", 0);
       }
    }
    return(value);
}
```

.info File Format

The .info file has the following format:

# comments
:*keyword1* [*keyword2* [*keyword3*]]
message text

You can include comment lines in your .info files by preceding them with the number sign. Use an initial colon to denote a line containing a keyword or keywords. If several keywords pertain to the same help message, place them on the same line, with spaces separating them. The message text supplied appears in the Spot Help window whenever this .info file and *keyword1*, *keyword2*, or *keyword3* are values for the HELP_DATA attribute.

:*keyword4*[:*file* [*page#*]]
message text

You also can specify another file, and optionally, a page within that file. Spot Help then locates the file specified and displays information on that page whenever this .info file and *keyword4* are values for the HELP_DATA attribute. This format produces the same result as a hypertext link, described on pages 88–89.

:*keyword5*[:{key}*file*:*keyword6*]
message text

The message text supplied appears in the Spot Help window whenever this .info file and *keyword5* are values for the HELP_DATA attribute. {key} indicates a hypertext link to *keyword6* within the file specified (see pages 88–89).

Generally there is a one-to-one relationship between keywords and help messages. If anything follows the keyword on the same line, it usually is the name of a file, preceded by a colon. When a file name is present, as on the :*keyword4* line in the second example, the More Help button appears on the Spot Help window (see Figure 6-2 on page 77). If a user selects the More Help button, the help mechanism displays the specified page of the referenced file; the default page number is 1.

The following example illustrates the first of these options, using Spot Help for the New Mail button in the mailtool application. The HELP_DATA attribute that defines the display is part of the null terminated attribute list, used to create panel items (the *SunView Programmer's Guide* describes this in detail). The entry in this attribute list is:

```
HELP_DATA, "mailtool:newmail"
```

When help is requested on this object, the Help mechanism finds the :newmail keyword in the mailtool.info file. It then displays the text lines appearing below the :newmail keyword as a Spot Help window, in this case as shown in Figure 6-2 on page 77. The entry associated with the newmail keyword in the mailtool.info file is as follows:

```
:newmail:mail/Receiving_Mail
    New Mail button


   Retrieves your new mail messages and
   makes permanent any tentative message
   deletions and message text changes.


   Click right: Provides the option to Com-
   mit changes without retrieving new mail.
```

Because the :newmail keyword is followed by a colon and a file name (the Receiving_Mail file in the default help directory) the Spot Help window includes a More Help button. If the user clicks left on this button, Help Viewer displays the page shown in Figure 6-3 on page 78.

sun_external.info Keyword File

If you refer to a topic in one of the handbooks provided on the Sun386i system, use the sun_external.info keyword file. Using sun_external.info protects you from any future reorganizations of these topics. For example, if Sun renames a file that tells users how to scroll windows, the change will appear in the keyword file sun_external.info, and the file renaming will not affect you.

For example, suppose that your application has a scroll bar, with a keyword of :scrollbar. If you want the More Help button to display the Scrolling Windows topic in the *Desktop Handbook*, your .info file entry would be similar to:

```
:scrollbar:{key}sun_external:desk_scroll
```

followed by your message text.

Then when a user presses the More Help button on your scroll bar Spot Help window, the Help Viewer:

1. Goes to the sun_external.info file.
2. Finds the :desk_scroll keyword.
3. Finds the sunview/Scrolling_Windows 1 path name and page number associated with that keyword.
4. Displays the Scrolling Windows topic in the viewer.

The {key} indicates a keyword link. Links are described in the Hypertext Overview section, starting on page 88.

Viewing Your Help Text

To view one of your Spot Help messages:

1. Save the .info file.
2. Restart your application.
3. Place the mouse pointer over the object that you want to check.
4. Press [Help].

If SunView cannot find a .info file, or a keyword within a . info file, a pop-up window appears explaining the situation.

Spot Help Guidelines

If you follow these guidelines, you will present users with Spot Help messages that are consistent with those supplied on the Sun386i system:

- Provide each Spot Help message with a title that identifies the help object as a button, menu item, window, and so on. Look at the Spot Help provided with Sun386i applications for terminology.
- Put the title on the line immediately below the keyword, indent it three spaces, and follow it with a blank line.
- Indicate what the object does (and how to make it function, if it is not obvious) briefly and precisely. Follow the message text with a blank line.
- Make Spot Help windows wider than they are tall (by a factor of two or so). You should make the typical 6- to 10-line message, including blank lines, 30 to 40 characters wide.
- When providing More Help references to topics in the standard Sun386i handbooks, look at the sun_external.info file first to see if the topic you want is listed. If it is, refer to it indirectly (as described on the preceding page).
- When providing More Help references to topics in your own handbooks, specify page 1. Opening a topic to its first page helps to orient users.

Help Viewer Interface

Help Viewer handbooks shipped with the Sun386i system were created using off-the-shelf "desktop publishing" software. You can choose from:

- Frame Maker™ (1.1 or later), available from Frame Technology Corporation (San Jose, CA 95131)
- Interleaf™ TPS (3.0.18 or later), available from Interleaf, Inc. (Cambridge, MA 02141)

You can mix and match documents from either source—Help Viewer can display either—so choose the one that best meets your needs. Templates for both packages are provided, to make entering help text easier. (The user handbooks Sun supplies on the Sun386i system were done with Frame Maker; the *Help Writer's Handbook* was done with Interleaf.)

The interface to Help Viewer is solely a writer's interface, not a programmer's. The Help Viewer facility merely displays any document it receives. Writers create documents for Help Viewer as they would any other document. That is, writers can use all of the text and graphics capabilities provided by Frame Maker and Interleaf TPS to create documents for on-screen display. The on-screen appearance of the documents during their creation is—when divorced from their Frame- or Interleaf-specific menus, borders, guides, and so on—exactly what the documents will look like in Help Viewer. The only difference faced by the writers is in the creation of hypertext links.

The rest of the discussion covers three main areas:

1. Help Viewer files and directories on the Sun386i system
2. Description and use of hypertext links, including how to create them in Frame Maker and Interleaf TPS
3. Writing Help Viewer handbooks, including how to use templates, where to put files, guidelines for writing handbooks, checking topics that you've written, adding your handbook to the Top Level table of contents so users can view it, and procedures to follow for installing your help files

Help Viewer Files

All standard help system files supplied with the Sun386i system are located in the default help directory, initially `/vol/help`. Administrators can change the location, and then use **Defaults** on the SunView Menu to tell the Help system where to look. Pages 79–80 provide an overview of the help directory structure.

System-supplied help files include:

- One text file, `/vol/help/language/USA-English/Top_Level`, that's the Top Level table of contents
- `.info` files in `/vol/help/language/USA-English`, containing Spot Help messages for the SunView Desktop and SunView applications
- Subdirectories containing Frame Maker and raster image (`.rf`) files for the standard Sun386i handbooks and Master Index `/vol/help/language/USA-English/`*handbook*
- A `Frame` subdirectory containing font files, initialization files, and template files used to create the standard handbooks in `/vol/help/format`
- A `help_guide` subdirectory (if you've loaded the `help_guide` cluster; see page 79) containing Frame Maker and Interleaf files for the *Help Writer's Handbook* in `/files/loaded/devel`
- An `Interleaf` subdirectory containing Interleaf files, fonts, and template files used to create the *Help Writer's Handbook* in `/vol/help/format`

The `.info` files and the corresponding handbook directories are listed, by handbook, in Table 6-1 on the next page.

Table 6-1 *Help Viewer Handbooks and Spot Help* `.info` *Files in* `/vol/help/language/USA-English`

Handbook	`.info` ***File***	***Directory***
Color Editor	`coloredit.info`	`coloredit/`
DOS Windows	`dos.info`	`dos/`
Help	`help.info`	`help/`
Help Writer's	—	`help_guide/`
Mail	`mailtool.info`	`mailtool/`
Master Index	—	`master_index/`
Organizer	`organizer.info`	`organizer/`
SNAP	`snap.info`	`snap/`
Desktop	`sunview.info`	`sunview/`
—	`sysex.info`	—
Text Editor	`textsw.info`	`textsw/`
—	`ttysw.info`	—

All help files, with the exception of the *Help Writer's Handbook*, come preloaded on the Sun386i system. To load the *Help Writer's Handbook*, follow the four steps shown on page 79.

If you look into one of the handbook directories, you will see that the document titles are the same as their displayed titles in Help Viewer except for the inclusion of underscores between words. For example, the file containing the Help Basics topic is named `Help_Basics`. This is done intentionally so that when topics are displayed in a topic history they will appear properly. Help Viewer strips out the underscores and replaces them with spaces. (You can display a topic history by pulling right on the Go Back option, available on the Help Viewer menu.) For this reason, it's a good idea to follow the same file-naming convention.

Hypertext Overview

At the simplest level, a hypertext link is a connector that leads from a particular place in one document to a particular place in another. Hypertext links in an on-screen document set let readers navigate around the documentation hierarchy. This means that users can successively display documents by following the links provided; they can double click on link text or select the text and then choose the Follow Link option from the Help Viewer menu to follow a link. It is part of a writer's job to insert the appropriate links into the on-screen documents.

Hypertext links are implemented in the form of *markers* embedded in the text of a document. Since markers themselves are not visible to readers, a change in font or font characteristic indicates the presence of link markers. Although any font or font characteristic can denote hypertext link markers, you should use underlining if you want to be consistent with the on-screen help provided with the Sun386i system. (You cannot use underlining alone to delimit hypertext links in Interleaf documents; see Creating Hypertext Markers in Interleaf Documents on page 93 for details). When a user follows a link, Help Viewer searches for the marker within the area delimited by the font change and then performs the action indicated by the text associated with that marker.

Currently, you can direct Help Viewer to do five basic things:

- Display a new document
- Display a pop-up window containing a document
- Change the page within a topic
- Execute a program or SunOS command
- Find a keyword in a file and execute the instructions associated with that keyword.

The types of links required to do each of these are described below.

Document links tell Help Viewer which file name to open for display; many of the markers in the standard set of Help Viewer handbooks supplied with Sun386i systems are for document links.

Pop-up links are similar to document links, but the associated help text appears in a temporary pop-up window that disappears as soon as a user presses a key or mouse button. Pop-up links are useful for displaying materials that might ordinarily be relegated to footnotes or for displaying more detailed information.

Program links instruct Help Viewer to do things such as change pages within the currently displayed Help Viewer topic. Page changes are not recorded in the topic history; if you want the topic and page number to appear in the topic history when a user follows the link, use a document link instead.

System links provide a vehicle for passing commands through to the SunOS system. You can use them to start interactive demonstrations, applications, or SunOS commands.

Keyword links specify a keyword in a keyword file, where the Help Viewer looks for instructions. Keyword links can resolve to any of the above four link types. Therefore, you can indicate a file to be displayed (by specifying a document, pop-up, or another keyword link) or a command to be performed (by specifying a program or system link). Use of keyword links makes it easier for you to maintain correct links when many markers point to an object that is likely to change.

Specifying Markers

This section describes how to specify markers for implementing each link type. The precise steps to follow for creating markers depend on whether you are using Frame Maker or Interleaf TPS. The two subsequent sections describe these steps.

Specifying Document Markers

In the simplest case, for a document marker, the marker text indicates the file name and optionally the page number of the new document, such as:

```
mailtool/Mail_Basics 2
```

This sample tells Help Viewer to display page 2 of the file `mailtool/Mail_Basics` in the default help directory (see page 80). If included, the page number must be separated from the file name by a space; if omitted, Help Viewer displays the first page of the named document.

Specifying Pop-up Markers

To construct a pop-up marker, perform the same steps as to construct a document marker, except precede the marker text with the special instruction `{pop}`. This instruction tells Help Viewer to place the specified text in a pop-up window. Therefore, the text:

```
{pop}mailtool/Mail_Basics 2
```

directs Help Viewer to display page 2 of the file `mailtool/Mail_Basics` in a pop-up window.

Specifying Program Markers

Program markers contain the instruction `{prog}`, followed by the Help Viewer command to be executed. There are five commands at present, all of which involve changing pages within the currently displayed topic:

```
{prog}PageNext
{prog}PagePrev
{prog}PageFirst
{prog}PageLast
{prog}PageGoto page_number
```

The page buttons at the top of every handbook page use the `{prog}PageNext` and `{prog}PagePrev` commands. While you could use document markers for the same purpose, using program markers lets users change pages much faster since the Help Viewer doesn't have to reload the topic.

Specifying System Markers

To create a system marker, use the instruction `{sys}`, followed by the command to be executed. Enter the command exactly as you would type it into a C-shell. For example

```
{sys}textedit&
```

starts the Text Editor application. The string is passed to the C-shell via the SunOS `system`(3) call.

System markers extend the capabilities of the Sun386i Help facility, but you should be aware of problems that can occur when running programs requiring tty input and output. For example:

```
{sys}more filename &
```

can hang the Help Viewer indefinitely because the `more`(1) command prints out the first part of the file *filename* and then waits for input before continuing. Also, you can't be sure where the output from `more` will appear, nor where a user should supply input.

Although you can easily start a window-based application from Help (for example, `{sys}cmdtool &`), invoking a command such as `more` requires a little extra care.

The best way to run a shell command from Help is to start a cmdtool and execute the command there. The following example uses a Help link to start a cmdtool, where the more(1) command is then executed on *filename*:

```
{sys}cmdtool syswait "Press any key to continue."
'more filename'
```

The syswait(1) command (new with the Sun386i system) prevents cmdtool from disappearing as soon as more exits. When more finishes, syswait displays the Press any key to continue message. cmdtool disappears as soon as a user presses any key in the cmdtool window. As long as cmdtool is displayed, Help Viewer will not accept input or redisplay its window. If a user tries to perform either function, Help Viewer displays the hourglass cursor.

The on-screen Using System Links topic of the *Help Writer's Handbook* contains additional examples. The *Help Writer's Handbook* is included in the help_guide cluster of the Developer's Toolkit software. If this cluster is not already on your system, refer to page 79 for instructions.

Specifying Keyword Markers

Creating a keyword marker requires several steps, but can save time later on. To create a keyword marker:

1. Create a marker with the instruction {key}, followed by a file name and keyword; for example,

   ```
   {key}textedit:intro_to_scrollbars
   ```

   where textedit is the file name and intro_to_scrollbars is the keyword. A colon must separate the file name and keyword, and each keyword must be unique within a given file.
2. Create a text file (not a Frame Maker or Interleaf file) named *filename*.info (if it doesn't already exist). Using the above example, you would create the file textedit.info.
3. Open *filename*.info and include the location of text to be displayed or actions to take when Help Viewer encounters the keyword specified in step 1. In this example, the keyword is intro_to_scrollbars. Therefore, in textedit.info, you could include an entry such as:

   ```
   :intro_to_scrollbars:Tool_Basics 5
   ```

When Help Viewer sees{key} (indicating a keyword marker), it checks the specified file for the keyword, and then performs the action indicated. In the above example, Help Viewer would display page 5 of the Tool_Basics file. In other words, the keyword link set up in your file ultimately resolves into a document link. But, in the general case, it could also resolve into any of the other link types, even another keyword link. For instance, the :intro_to_scrollbars keyword could be associated with a system marker, in which case the specified command would be performed. If this were a system marker that started the Text Editor application, the syntax would be:

```
:intro_to_scrollbars:{sys}textedit&
```

If you use keyword markers to provide a number of links to the same description, and the location of that description changes in a future revision of your handbook, you only have to alter the `.info` file, not each of the markers.

Use of the `sun_external.info` file is another example. This file, located in `/vol/help/language/USA-English`, provides a keyword interface to many of the basic subjects covered in the standard Sun386i handbooks. Suppose, for example, that you wanted to create a link to the Scrolling Windows topic in the *Desktop Handbook*. To use the keyword interface, the marker in your document would be:

```
{key}sun_external:desk_scroll
```

This identifies the keyword link type, the `sun_external.info` keyword file (the `.info` extension is assumed), and the `:desk_scroll` keyword. If you look for this entry in the `sun_external.info` file, you will see:

```
:desk_scroll:sunview/Scrolling_Windows 1
```

The instruction associated with this keyword, `sunview/Scrolling_Windows 1`, is a document link. All of the keywords in this file are associated with document links to topics in the standard set of Sun386i handbooks.

Creating Hypertext Markers in Frame Maker Documents

Before using Frame Maker to create hypertext markers, you should do a small customization to make marker identification within Frame Maker easier:

1. Open the text file `$FMHOME/.makerinit/markers`. (If you do not have read/write access to the file, become superuser by issuing the `su` command.)
2. Change the string `Type 9` to `Hypertext-out`.
3. Save the changes and exit the file.

These steps modify Frame Maker so that the Edit Markers dialog now includes the Hypertext-out marker type instead of Type 9 in the unmodified program.

To insert a hypertext marker in a Frame Maker document:

1. Set the insertion point for the marker; this should be in text that is or will be a visible link for users (denoted by either a different font or a different font characteristic; the convention used on the Sun386i system is underlining).
2. Select the **Markers...** option from the Frame Maker Edit menu.
3. Select the **Hypertext-out** marker type.
4. Enter the text of the marker.
5. Click left on the **New Marker** button.

To edit a hypertext marker, select it, make the edits in the Markers dialog, and click left on the **Edit Marker** button.

Creating Hypertext Markers in Interleaf Documents

To create hypertext markers in Interleaf documents, you must have Interleaf TPS 3.0.18 or a later version. (Contact Interleaf for an upgrade to revision 3.0.18 if you're running an earlier version.)

To insert a hypertext marker in an Interleaf document:

1. Set the insertion point for the marker; this should be in text that is or will be a visible link for users (denoted by either a different font or a different font characteristic; the convention used on the Sun386i system is underlining).
2. Pop up the document menu, pull right on the **Create** option, and select **Index**.
3. On the Index properties sheet, while in Format mode, enter **Hypertext-out** in the Level 1 field.
4. Enter the text of the marker in the Level 2 field.
5. Switch to Custom mode and enter **Comment** in the Index Document field.
6. Click right for the menu and select **Apply**.

To edit a hypertext marker, select it, pop up its property sheet, make the edits, and select **Apply**.

Because underlining in Interleaf does not change the font, you must manually change the space or punctuation characters on both ends of the underlined text for link highlighting to work properly in Help Viewer:

- If there are space characters, use "hard" spaces (for instance, press Esc and then the space bar).
- Change the point size, weight, or typeface, or indicate italics.

Writing Help Viewer Handbooks

The templates you need to get started are in the following subdirectories:

- `/vol/help/format/Frame/templates`
- `/vol/help/format/Interleaf/templates`

The Frame Maker templates are based primarily on the user handbooks (such as the *Organizer Handbook*), and the Interleaf templates primarily on the *Help Writer's Handbook.*

To use a template, first copy it to your working directory. While this step is not required, it will help ensure that you don't inadvertently write over a template that should remain as is. Then open the template and rename it to the name of your topic. If the name of your topic is Counting to Twenty, for example, then name its file `Counting_to_Twenty`. That is, substitute underscores for spaces and don't use any extensions. This is necessary for the Help Viewer's topic history to display properly.

Every paragraph in a Frame Maker or Interleaf document is formatted according to a particular set of specifications. Which set depends on the paragraph's *tag* in the case of Frame Maker or *component* in Interleaf.

The chief value of the template documents, aside from providing you with proper page sizes and margins, is in the predefined sets of tags and components they include, as described below:

- **`first` or `fst` suffixes** — Several tags and components have the suffix `first` or `fst`, for example, `bullet_first` and `para_first` in Frame, and `bul.fst` and `para.fst` in Interleaf. Any such component is meant to be the lead paragraph in a new section. A new section is denoted by a left marginal head and has extra space above, which is the purpose of these components. (In Interleaf, change any `para.fst` that appears as the first component on a new page to `para.top`.)
- **`px` or `pix` suffixes** — Components ending in `px` or `pix` are intended for use in front of graphics frames, for example, `bul.px`.
- **`bullet_last`, `list_last`, `bul.lst`, and `list.lst`** — Use `bullet_last` and `list_last` (Frame) or `bul.lst` and `list.lst` (Interleaf) to end lists, whether or not these components are followed by graphics.

Handbook Guidelines

The guidelines that follow assume that you are writing handbooks that will appear alongside the standard handbooks included with the Sun386i system. If you follow these guidelines, you will provide users with a consistent approach and appearance.

- Use the templates provided with the Sun386i system. This is the single most important way of ensuring consistency with Sun handbooks.
- Relegate conceptual (narrative) material to the "basics" topics, and restrict the "how-to" topics to procedural material.
- Limit "basics" topics to less than ten pages and procedure topics to less than five. If it can be so divided, two three-page topics are preferable to a single six-page topic.
- Write in a modular fashion, making each topic and each conceptual or procedural section within a topic as self-sufficient as possible.
- Use in-text hypertext links sparingly. If you write modularly, lots of cross references shouldn't be necessary.
- Use graphics generously.

Adding Help Files to the Default Help Directory

As shipped, `/vol/help` is the default directory for all Spot Help and Help Viewer files. When you are developing help for your application, however, you should create your own default help directory (possibly under your user directory), copy the contents of `/vol/help` to that directory, and then use Defaults on the SunView Menu to specify this directory as the default for the Help system to use.

The Spot Help file for your application must be in the default help directory that you specify with Defaults, and it must have the same name as the program it documents. Similarly, the subdirectory containing the files for your application's handbook should be in this directory and have the same name as the program it describes. To recursively copy the links in the directories below `/vol/help` to the directory chosen, you can use the command format `cp -r` *new_default_help_directory* from within the `/vol/help` directory.

If you are using Frame, you can work on handbook files in the default help directory. If they have the appropriate links and the handbook contents page is in the Top Level (also in the default help directory), you can look at them at any time in the Help Viewer.

If you are using Interleaf, your working files will be in your `desktop` directory; you will have to transfer them to the default help directory before displaying them in the Help Viewer. However, first you must convert them to Printerleaf format by performing the steps below.

1. Pop up the Printer menu.
2. Pull right on **Printerleaf** and select **Document**. This saves the file in Printerleaf format and appends the `.pl` extension to its name.
3. When you move, copy, or rename the file in the default help directory, strip off the `.pl`.

Checking Topic Appearance and Function

As you create topics you will likely want to check their appearance and the functioning of hypertext links in the Help Viewer. To do this:

1. Copy your handbook files into the default help directory that you've specified with Defaults (if they are not already there).
2. Add your handbook to the copy of the `Top_Level` file in the default help directory so that you can link to it from the viewer's Top Level table of contents.

The following section discusses the second step.

Adding Handbooks to the Top Level

No one can access your handbook from the Help Viewer until you insert the handbook title in the Help Viewer's Top Level table of contents. `Top_Level` is a special text file (not a Frame or Interleaf file) in a subdirectory of the default help directory that you can edit for this purpose. `Top_Level` is initially in `/vol/help/language/USA-English`.

The file lists the standard set of Sun386i handbooks as a series of entries. Each entry consists of a text string in single quotes on the left and a path name and page number in square brackets on the right. The strings are displayed as underlined contents items; the path names are links to the files displayed after following the link.

You edit `Top_Level` as you would any other text file. The only things to note are the use of:

1. Backslash characters in front of apostrophes, or backslashes that are part of your handbook's title
2. Number signs in front of comments

The next section describes installing your help files, and includes suggestions for appending your handbook entry to `Top_Level` via a shell script.

Installing Your Help Files

This section suggests the steps that an installation script should perform to load help for your applications, as well as the steps that installation instructions should include to ensure that your help is easily available to all users on a network.

Adding to the Top Level During Installation

The Top Level might already be customized to a certain extent. Therefore, it's suggested that you write a shell script that appends your handbook entry to the `Top_Level` file in `/vol/help.master` (a duplicate of `/vol/help`) when a customer installs your application. Such a script would resemble the following:

```
echo "'application_name Handbook'
[application/application_name_Handbook]" >>
/vol/help.master/language/USA-English/Top_Level
```

Or, to check for the presence of your handbook before adding it:

```
if grep 'application_name Handbook' Top_Level
then
  echo "'application_name Handbook' is already
   installed in 'Top_Level'

else
  echo "'application_name Handbook'  [application/
   application_name_Handbook]" >>
   /vol/help.master/language/USA-English/Top_Level
  echo "'application_name Handbook' installed in
   'Top_Level'"
fi
```

Making Your Help Easily Accessible on a Network

As described on page 80, the default help directory is `/vol/help`. It is recommended that administrators retain this as the default directory for easier maintenance, and for disk space conservation and network-wide availability of Sun386i and third-party help files. For the Help system to access your files through `/vol/help`, you must add a few steps to your installation script, and must instruct system administrators to perform certain steps after installing your software.

Installation Script Steps

As part of the installation procedure, your script should:

1. Create the subdirectories *application_name*`/language/`*language*`/help` for Spot Help files and *application_name*`/language/`*language*`/help/`*handbook* for handbooks in `/usr/local` if there's room (see pages 144-145). *language* can be either:

   - `USA-English`
   - `English`
   - `French`
   - `French_Swiss`
   - `German`
   - `German_Swiss`
   - `Italian`
   - `Swedish`
   - `Spanish`

   Then add your help files to these directories. (See pages 206–207 for a description of the suggested directory structure for applications.)

2. In the directory `/vol/help.master`, create two link files. The first one shown on the next page is for `.info` files:

`ln -s` *filename linkname*

where *filename* is

`/vol/`*application_name*`/language/`*language*`/help/`*application_name*`.info`

and *linkname* is

`/vol/help.master/language/`*language*`/`*application_name*`.info`

The second link file, for handbooks, has the same format but *filename* is

`/vol/`*application_name*`/language/`*language*`/help/`*application_name/handbook*

and *linkname* is

`/vol/help.master/language/`*language*`/`*application_name*`.info`

Installation Instructions for System Administrators

If your installation script performs the steps just described, or if you're adding Organizer icons for your application's files, whoever loads your application **must perform** the steps below to access your help or icons. Although these instructions also appear in *Sun386i Advanced Administration,* it's a good idea to include them in the written installation instructions that you provide with your application.

1. Export the application so that anyone on the network can access it.
   a. Create a symbolic link in the command format:

      `ln -s` *location_of_application* `/export/local/`*application_name*

      The suggested location for applications is in subdirectories under `/usr/local/`*application_name* (on Sun386i systems, this is a link to `/files<`*n*`>/local`), if there is room; see page 202.

   b. Become superuser by entering **su** and the superuser password.
   c. Export the link just created by adding to the `/etc/exports` file, which lists all exported directories. Use the format:

      `/export/local/`*application_name* `-ro`

      The `exports`(5) `man` page and *Sun386i Advanced Administration* provide details about this file and exporting.

   d. Run `exportfs -a` to make the mount daemon aware of the change to `/etc/exports`. (See the `exportfs`(8) `man` page for details.)

2. Make the application accessible from a volume (see page 205) by updating the `auto.vol` file on the Yellow Pages master.
   a. Enter `rlogin 'ypwhich -m auto.vol'` to log in to the Yellow Pages master.
   b. Become superuser by entering `su`.
   c. Create a volume for the application by adding an entry to the `/etc/auto.vol` file in the format:

      application_name *system*`:/export/local/`*application_name*

   d. Rebuild `auto.vol` by issuing the commands:

      ```
      cd /var/yp
      make
      ```

After a system administrator performs these steps, your help files will be available to any user on the network. You might want to follow these steps yourself after your help files are completed, to ensure that everything is working correctly. If you do, be sure to move your files to `/vol/help/language/`*language* (page 96 lists values for *language*), and use Defaults on the SunView Menu to change the Help directory to `/vol/help`.

6.3 Administration Facilities

The ease-of-use administration facilities described in this section are:

- `snap`(1)
- Automatic System Installation
- New User Accounts

All of these facilities are built on top of two new SunOS 4.0 features: secure RPCs (Remote Procedure Calls) and the Yellow Pages (YP) updater program. Secure RPCs use a public key encryption technology that provides user authentication in the network. The YP updater provides a way to update YP maps from programs. To use these features, your network must be running the Yellow Pages and a Sun386i system must be the YP master. *Sun386i SNAP Administration* describes RPCs and the Yellow Pages in detail.

The next three sections present an overview of the administration facilities on the Sun386i system. For details, see *Sun386i SNAP Administration.*

The snap Program

`snap`(1) provides a window interface for users with the required privileges to browse, add, delete, and modify user accounts, user groups, systems, modems, terminals, and printers. When a user confirms changes made, `snap` makes the appropriate changes to the Yellow Pages facility and directories. `snap` privileges are implemented through membership in special groups. As shipped, `snap` gives all users all `snap` privileges; most sites probably will want to restrict privileges to some degree.

`snap` also enables:

- A window-based method for selecting and installing clusters that provides an alternative to using the `load`(1) and `loadc`(1) commands
- A window-based method for installing third-party software put on the release tape or diskette with the `bar`(1) command (see page 145 for details)
- File backup and restore—The Sun386i system has a backup facility that provides easy-to-use personal and system-wide backup functions. Both full and incremental backups are possible. Users can also restore files with `snap`.

Automatic System Installation

Using Automatic System Installation, a user can add a system to an existing Sun386i network in about 30 minutes after unpacking. The first time a user powers on the Sun386i system, one of the following four scenarios occurs, depending on the machine's configuration.

For a standalone, the system asks for confirmation that it is not going to be added to a network, and then continues with the boot procedure.

For a Sun386i system establishing a network (YP master), the system asks for confirmation that it should set up this system as a YP master and then continues to do so.

For a system attaching to an existing Sun386i system network, the system is automatically configured onto the network, enabling users with appropriate privileges to access files on the network. This applies to both diskful and diskless systems, provided the availability of a diskful system to act as a boot server if a diskless system is to install itself. (The diskful system would need about 25 Mbytes of free space, as well as `/usr` binaries for the diskless machine.)

For a diskful system attaching to an existing network that is not a Sun386i system network, the process is more complicated. The system requests confirmation that the network is not an Automatic System Installation (Sun386i system) network. Then an administrator must add the system using procedures documented in *Sun386i SNAP Administration.* (These are the same procedures that administrators must follow if they disable Automatic System Installation via `snap`.)

New User Accounts

New User Accounts consists of two parts:

- New user access, which displays directions for creating a user account (for users who don't have a log-in name on a system). A log-in name and resources such as a home directory are established. Administrators can turn off this ability with `snap`.
- Full screen login, whereby users entering their user names and passwords can view help screens by pressing the [Help] key. When users press [Help], detailed instructions on how to log in are displayed. (This help is independent of the Spot Help facility; the Spot Help description begins on page 76.)

Because of the log-in screens and help facility, the above processes run only on standard Sun386i bitmapped displays. By default, each system ships with the `getty -n` command in the `/etc/ttytab` file to enable display of New User Accounts screens. To disable these screens, remove the `-n` option from the `/etc/ttytab` file.

6.4. Using Color

Color is an important part of the Sun386i system. This section describes the user and programmer tools that facilitate the use of color in applications. The first part of this section provides basic information on the use of color. For additional information, refer to the *SunView Programmer's Guide*. Also, the *SunCGI Reference Manual* describes color functions and color attributes affecting the display of output primitives.

SunView Color Basics

Each point (pixel) on a color monitor represents an 8-bit value. This value is used as an index into a *colormap*. Each colormap entry contains a 24-bit value, 8 bits for each of the three primary colors: red, green, and blue. The number of possible colors, therefore, is 2^{24} or approximately 16 million. The number of colors that can be in use at any one time, however, is limited to 256.

Setting the colormap—establishing the correspondence between 8-bit and 24-bit pixel values—for use by an application is the responsibility of the application programmer. Actually, you do not set the colormap directly, but rather a *colormap segment* for the particular application. The colormap is a lower-level hardware resource, encompassing all of the colors available for display across all window applications running concurrently. The colormap segment is the vehicle by which an application specifies the particular set of colors it needs at any given time.

The colormap segments for all the currently running applications together form the current colormap. If the sum of the current colormap segment entries for current processes exceeds 256, the colormap limit, then swapping occurs. The window system gives priority to the colormap segment of the window under the mouse pointer, and swaps out segments belonging to other windows. You can share colormap segments among windows and processes, or you can prevent them from being shared if you are using them dynamically (e.g., for animation). If a colormap segment's use is static, then you should use a shared colormap segment definition to decrease the potential for swapping. The *SunView Programmer's Guide* provides a more complete description about color and colormap segments.

Foreground and Background Colors

The most basic use of color in a SunView application is to set the foreground and background colors of its frame. If you look at a typical SunView application like the Text Editor window, the foreground color is the one used for the borders and any displayed characters. The background color is just that, the background in the character display area of the window. The colors used in the namestripe at the top of the window are reversed, with the background color being used for display of characters on the foreground-colored border.

The foreground and background colors for an application are determined by the last and first values, respectively, in the application's colormap segment. It is not necessary to directly manipulate the colormap, however, in order to use foreground and background colors. Two other interfaces exist, one at the programmer level, the other at the user level.

The first is provided by the attributes `FRAME_FOREGROUND_COLOR` and `FRAME_BACKGROUND_COLOR`, which are set when you create a program frame. A related frame attribute is `FRAME_INHERIT_COLORS`, a flag specifying whether the frame's subwindows get the same foreground and background colors as the frame.

The user-level interface is provided by the `coloredit`(1) facility, described on page 104. In addition, there are several command-line frame arguments corresponding to the three frame attributes described above. They are, `-Wf [r] [g] [b]`, `-Wb [r] [g] [b]`, and `-Wg`, respectively specifying the foreground color, background color, and which subwindows inherit these colors. `r`, `g`, and `b` are the intensity values (0–255) for the RGB components.

You can use these arguments:

1. When invoking a SunView application directly from a `shelltool` or `cmdtool` window
2. In the `.sunview` (called `.suntools` prior to the SunOS 4.0 system) or equivalent file executed when `sunview` is invoked
3. In the `/usr/lib/rootmenu` or equivalent file that specifies the items on the SunView rootmenu, and the corresponding commands executed when a user selects rootmenu items

Note that application-specified frame attributes override user-specified command-line arguments, regardless of where the latter originate. If the foreground and background colors are specified neither at the program nor user levels, they are derived from SunView defaults. There are two levels of defaults for window-based applications:

- User-specified red, green, and blue values supplied with the `-f` and `-b` options to the `sunview` command (see the `sunview(1) man` page)
- The default colormap segment defined in `<sunwindow/cms_mono.h>`, if `sunview` is invoked without the `-f` and `-b` options

The `-f` and `-b` options specify the foreground and background colors (RGB values) for the SunView root screen, and for all pop-up menus as well. They also are used to specify foreground and background colors in applications for which these colors are not otherwise specified.

If `sunview` is invoked without the `-f` and `-b` options, then the defaults are taken from the default colormap segment defined in `<sunwindow/cms_mono.h>`. This is the monochrome colormap segment, and specifies a black (0 0 0) foreground on a white (255 255 255) background. (The `sunview` inverse video switch, `-i`, swaps the foreground and background colors.)

Panel Colors

You also can set colors for panels and panel items by dynamically manipulating colormaps. This section describes the various types of panel items and explains how to set colors for them. The method for adding color to panels is very similar to that for adding color to canvases. The *SunView Programmer's Guide* describes adding color to canvases, and provides additional background information.

Panels are comprised of *items* that users can choose to perform various functions. The six basic types of panel items are listed and briefly described below.

- Message items—descriptions, pictures, and dynamic status messages that users can view by selecting message labels; a label can be an image or a string in a specified font.
- Button items—labels that initiate commands when selected; buttons have visible feedback for previewing and accepting selections made.
- Choice items—a list of selections for users, one of which is the *current* choice; choices are either fully displayed or require user interaction to view all available choices.
- Toggle items—a list of elements that behave as toggles; each choice is either on or off, independent of other choices, and selecting a choice changes its state.
- Text items—labeled fields that users fill in; optionally, clicking right on text items can produce a menu from which users can select. Your application can process user input on a per character, per field, or per screen basis.
- Slider items—graphical representations of a value within a range; the `coloredit(1)` program provides sliders for adjusting the hue, saturation, and luminosity of foreground and background colors. (`coloredit` is described on page 104.)

You also can create panels that are larger than the subwindows in which they appear by attaching scroll bars to them. The steps to add color to a panel differ depending on whether or not the panel has scroll bars. The following sections explain how to add color to both types of panels.

Adding Color to Nonscrollable Panels

To add color to a panel that does not have scroll bars:

1. Add the PANEL_ITEM_COLOR attribute to the code for each panel item that you want to color.
2. Use the WIN_PIXWIN attribute on the panel handler to get the pixwin of the panel.
3. Change the panel's colormap, using the pw_setcmsname() and pw_putcolormap() functions.

PANEL_ITEM_COLOR is a panel item attribute that is followed by an index into the colormap associated with this panel item. The index has a value type of int. panel_get and panel_set functions are valid with this attribute.

Manipulating the Panel Colormap

In addition to adding the above attribute to panel items, you also must name and set the size of the panel's colormap segment by performing the following two steps:

1. Name the colormap segment with pw_setcmsname().
2. Set the size of the segment by loading the colors with pw_putcolormap().

It is important to do both of the above steps in the order shown. The pw_setcmsname() function resets the colormap segment to a NULL entry. After setting the name, you must immediately call pw_putcolormap() to set the size of the colormap segment and load it with the colors desired. The calling conventions for pw_setcmsname and pw_getcmsname are shown below.

```
pw_setcmsname (pw, name)
    Pixwin *pw;
    char    name[CMS_NAMESIZE];
pw_getcmsname (pw, name)
    Pixwin *pw;
    char   name[CMS_NAMESIZE];
```

Adding Color to Panels with Scroll Bars

The view pixwin is a region of the pixwin that pertains to a scrollable panel. Therefore, to add color to a scrollable panel, set two colormaps: one for the view pixwin and one for the panel pixwin. When adding color to scrollable panels, attach the scroll bars to the panel *after* you change the panel's colormap segment, using the steps below.

1. Create the panel without scroll bars, as described in the preceding section.
2. Use the PANEL_PAINT_PIXWIN attribute to get the view pixwin of the panel.
3. Change the colormap of the view pixwin of the panel.
4. Change the colormap of the panel pixwin of the panel.
5. Attach scroll bars to the panel.

When specified in a window_get call, PANEL_PAINT_PIXWIN is a panel attribute that returns a pixwin pointer to the view pixwin of the panel. You cannot use window_set to change the view pixwin.

If a panel already has scroll bars, change the panel and view pixwin colormaps and then reattach the scroll bars to ensure that the scroll bar pixwin regions use the new colormap segment.

Color Panel Examples

This section contains two examples of adding color to panels. The first illustrates making a panel called CMS Name: blue against a white background:

```
cmd_panel_init(frame, panel)
Frame    frame;
Panel    *panel;
{

/* setup for color panel item test */

#define  WHITE   0
#define BLUE     1
#define  RED     2
#define BLACK    3

unsigned char    r[4],g[4],b[4];
char             *CmsName = "CPTest";
Pixwin           *pw;

/* creating panel */
    *panel = window_create(frame, PANEL,
                  WIN_RIGHT_OF, scroll_panel,
                  WIN_Y,0,
                  WIN_HEIGHT,(int)window_get
                  (scroll_panel,WIN_HEIGHT),0);
    cmd_panel = *panel;
/* creating and loading colormap */
        r[0] = g[0] = b[0] = 255;   /* white */
        r[3] = g[3] = b[3] = 0;     /* black */
        r[1] = g[1] = 0; b[1] = 255;/* blue */
        r[2]=255;g[2] = b[2] = 0;   /* red */

/* setting color for canvas colormap */
        pw = (Pixwin *)window_get(cmd_panel,
        WIN_PIXWIN,0);
        pw_setcmsname(pw,CmsName);
        pw_putcolormap(pw,0,4,r,g,b);
/* creating color panel item (CMS NAME) */
   cms_name_label_item =
         panel_create_item(cmd_panel,
         PANEL_MESSAGE,
PANEL_ITEM_COLOR, BLUE,
PANEL_ITEM_X,PANEL_CU(50),
PANEL_ITEM_Y,PANEL_CU(0)+4,
PANEL_LABEL_STRING, "CMS Name:",
0);
```

The second example shows how to attach color to a scrollable panel. The scroll bars will appear in the foreground color of the colormap.

```
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <sunwindow/cms_rainbow.h>

init_color_panel(base_frame)
    Frame base_frame;
{
    Panel          panel;
    Pixwin         *pw;
    unsigned char   red[CMS_RAINBOWSIZE];
    unsigned char   green[CMS_RAINBOWSIZE];
    unsigned char   blue[CMS_RAINBOWSIZE];
    panel = window_create(base_frame, PANEL, 0);
    cms_rainbowsetup(red, green, blue);
/* set the WIN_PIXWIN colormap */
    pw = (Pixwin *) window_get(panel, WIN_PIXWIN);
    pw_setcmsname(pw, CMS_RAINBOW);
    pw_putcolormap(pw, 0, CMS_RAINBOWSIZE, red,
    green, blue);

/* set the PANEL_PAINT_PIXWIN colormap */
    pw = (Pixwin *) window_get(panel,
    PANEL_PAINT_PIXWIN);
    pw_setcmsname(pw, CMS_RAINBOW);
    pw_putcolormap(pw, 0, CMS_RAINBOWSIZE, red,
    green, blue);

/* now attach scroll bars */
    window_set(panel,
        WIN_VERTICAL_SCROLLBAR,
       scrollbar_create(0),
        WIN_HORIZONTAL_SCROLLBAR,
        scrollbar_create(0),
        0);
```

The coloredit Program

Anyone with a color system can add or change window and icon colors with coloredit(1), and then save choices made with the toolplaces(1) command. You also can use coloredit as a prototype tool, to see what colors look best with your application. This section briefly describes coloredit. For more information about the user interface, refer to *Sun386i Advanced Skills*.

Users can select items for which they want to add or change color with the coloredit panel. Users can edit colors either by:

- Selecting a color from a scrollable subwindow of alphabetized color names

- Changing hue, saturation, and value settings
- Changing red, green, and blue (RGB) settings
- Using a combination of the previous three methods

The color names that appear in the `coloredit` panel are stored in the file `/usr/lib/.rgb`. Users can copy this file to their own directories and add to it. The colors added then appear in the scrollable subwindow of the `coloredit` menu.

The hue slider represents the 360 degrees of a color wheel. Changing the hue setting is analogous to changing the color of a filter on a stage light, for instance from red to green. Saturation represents the levels of the color chosen with the hue slider. It is analagous to using a different shade of the same filter, such as replacing a lighter shade of blue with a darker blue. The value slider represents luminosity; using the stage light analogy, increasing the luminosity represents increasing or decreasing the intensity of the light.

The hue, saturation, and value settings and the RGB settings can be used interchangeably. These two different methods permit the same fine tuning of color selections. As the setting of one slider is changed, the counterpart slider in the alternate set also changes to reflect the modification. However, only the RGB settings have physical representations. The numbers displayed beside the RGB sliders are the physical RGB values for the choices made.

The Color Index, located beneath the RGB sliders, specifies the colormap location of the color currently being edited. For a two-color SunView object, when the Color Index is 1, the foreground color (displayed by the Sun logo in the proof canvas) is being edited; when it is 0, this indicates background color editing, and the Sun logo disappears. To edit a different color, choose that color from the palette. The proof canvas then reflects the new color selected. For applications that have more than two colors, the Color Index for the background is still 0 but the foreground Color Index will be the size of the colormap (number of colors) minus one. That is, for a five-color application, the foreground Color Index is 4.

Using `coloredit` as a Development Tool

You can use `coloredit` to see which colors look best for your application. After you are satisfied with your choices, record the RGB values of each color used and then include those values for the applicable window item in your code.

Application Guidelines

This section provides some guidelines on the effective use of color and a list of standard colors with their corresponding red, green, and blue (RGB) values. General guidelines are presented first, followed by guidelines that are more specific to the Sun386i system; the latter are most relevant to applications intended to run alongside the standard SunView applications (such as Mail and Text Editor) that are part of the Sun386i system.

General Color Guidelines

Design your interface in black and white first. Use color as an added cue to distinguish different kinds of objects or information, not the only or primary cue. This avoids disfranchising color-blind users of your application and is in any case good practice.

Use colors sparingly. Color is appropriately used for directing attention to novel objects, for example, but overuse decreases its effectiveness.

- Applications that are primarily text-oriented should limit color use to foreground and background colors for their frames, and to occasional coloring of special areas or text.
- Applications that are primarily graphics-oriented should use black for the frame foreground (borders) and a neutral gray in areas where color objects are displayed. (Research has shown that color looks best against neutral gray.)
- Graphics applications intended for casual users should use a maximum of four colors.
- Graphics applications intended for experienced, long-term users should use a maximum of seven colors.
- As the number of colors increases, increase the size of the color-coded objects.
- Don't overdo color highlighting in the temporal dimension (for instance, yellow alerts every 30 seconds or so quickly lose their punch and become annoying.)

Use bright colors even more sparingly. Red, yellow, orange, and green readily attract the eye; overdoing it is distracting.

Use colors in a culturally consistent manner. When color-coded information has positive or negative connotations, reflect those connotations in your color choices. For example, in a financial chart showing losses and profits, use red for the losses, not the profits.

Select compatible color combinations. Avoid red-green, blue-yellow, green-blue, and red-blue pairs.

For character-background pairs, go for high contrast. Applications that are primarily text-oriented should use black on white or off-white (for example, wheat—look at `/usr/lib/.rgb` for RGB values). If you're dealing with a little text in a basically graphical application, use complementary character-background colors. The primary complementary pairs are blue-yellow, red-cyan, and green-magenta.

Sun386i System Color Guidelines

End users can easily alter window foreground and background colors with the `coloredit(1)` utility, described in the previous section on page 104. Therefore, unless your application makes use of color graphics for which you need to control the background or it is the only application available, don't supply foreground and background color frame attributes. Take the Sun386i system defaults and let end users set their own.

Also, the Sun386i systems are configured to come up in SunView when first booted (although you can change this, as can users). Because studies have shown that a background of medium gray is the best against which to view colored objects, the root screen will come up in a gray color (implemented by the `-pattern gray` option for `sunview`). Also, the root screen (and therefore pop-up menus) foreground colors are set to black and white (or off-white). Keep this in mind when adding color to your applications.

7

MS-DOS Environment

7

MS-DOS Environment

This chapter describes the MS-DOS environment—its features as a user interface and as a cross-development environment for MS-DOS. The chapter also discusses issues that you should be aware of if you port PC applications to the Sun386i system. For more information about customization and the MS-DOS user interface called DOS Windows (`dos`(1) is the name of the program), refer to the *Sun386i User's Guide* and *Sun386i Advanced Skills*. The latter manual contains more details about much of the information in this chapter. In addition, the *Sun386i DOS Reference Manual* provides information about most MS-DOS commands.

7.1. MS-DOS Overview

The Sun386i system runs MS-DOS 3.3. Briefly, MS-DOS and `dos`(1) let you:

- Run most PC programs without modification
- Run multiple PC windows
- Run most PC applications much faster than on any PC
- Run PC text-only applications that do not attempt to address the cursor, clear the screen, or display graphics (such as `BACKUP`, `CHKDSK`, `DIR`, and `RESTORE`) in a `cmdtool` or `shelltool` window, rather than only in 80-column by 25-line DOS Windows
- Run compilers, linkers, and the SunOS `make`(1) command on MS-DOS target files in `cmdtool` or `shelltool` windows by specifying those targets as text-only
- Decide which of the following three PC display adapters will be emulated: Monochrome Display Adapter (MDA), Color Graphics Adapter (CGA), or Hercules Graphics Adapter (the default)
- Emulate the Microsoft® Mouse or use the mouse as it functions under SunView
- Access an 8087 numeric coprocessor, emulated via the 80387 numeric coprocessor
- Configure DOS window options via the `setup.pc` file (described on page 114)
- Assign any SunOS directory its own drive letter to organize data and simplify references to paths
- Set up alternate names, called links, between MS-DOS and SunOS files
- Transfer data (Copy and Paste) between windows

- Share piped output between MS-DOS and SunOS commands
- Share MS-DOS files across a Sun386i network
- Perform piping into any PC application
- Use the `EDITDOS` program to edit an MS-DOS file in a `textedit` window, without first converting the file to SunOS text file format (the source code for this program starts on page 118)
- Use either of two standard SunOS screen fonts available to PC processes, `pcfont.r.14` (regular) and `pcfont.b.14` (bold), both in `/usr/lib/fonts/fixedwidthfonts`
- Install, allocate, and access boards plugged into the AT bus
- Convert between the PC and ISO fonts provided with the Sun386i system by using the `unix2dos`(1) and `dos2unix`(1) programs
- Convert text files for use by both operating systems with `unix2dos`(1) and `dos2unix`(1) programs

You can open DOS Windows either by typing **dos** or by selecting **Command Windows** from the root menu, pulling right, and selecting **DOS Windows**.

Additionally, if you add `setenv DOSLOOKUP on` to your `.login` file, you can implicitly open DOS Windows by entering the name of any MS-DOS program loaded on your system.

When you start DOS Windows, it automatically comes up with MS-DOS active and executes the `AUTOEXEC.BAT` file in the host machine's MS-DOS file area (Drive C:). BIOS code is loaded into memory from disk at this time. After completing the bootup procedure, DOS Windows is ready to accept user input for execution by the MS-DOS command processor. When you close DOS Windows, the SunOS system automatically closes all open MS-DOS files, flushes all buffers, and releases all assigned external devices and memory.

To enable enhanced graphics for PC applications, Sun386i users can select software emulation of the Monochrome Display Adapter (MDA), Color Graphics Adapter (CGA), and Hercules graphics cards. Alternatively, if the Sun386i system is hooked up to a PC, you can add one of these cards or an Extended Graphics Adapter (EGA) card to the Sun386i. You will see the enhanced graphics and increased graphics performance on your PC monitor.

An application running in DOS Windows behaves as if it were running on a PC, with access to the actual PC hardware. One difference is that compute-intensive PC applications run faster on the Sun386i system, due to the machine's speed and multitasking advantages.

7.2. Application Issues

This section contains information on memory, application naming, adding SunOS commands to your applications, text-only applications, running `make`(1) on MS-DOS targets, and file permission differences between MS-DOS and SunOS systems.

Memory

The Sun386i system allocates 640 KB of its virtual memory for each application running in DOS Windows. In addition, up to 2 Mbytes of expanded memory is available for each application that uses the Lotus®-Intel-Microsoft (LIM) expanded memory specification. Since the Sun386i system allocates virtual memory, no added hardware is required for this support. If you want to take advantage of expanded memory, you must design applications to access LIM memory by switching parts of the program in and out of LIM address space, which starts at the standard location `D0000`. LIM memory is available to each DOS Windows application that can use it, even if several windows are running simultaneously.

Naming Your PC Applications

Because MS-DOS and SunOS systems have different file-naming conventions, the SunOS system provides a file name mapping scheme that enables you to specify programs from within either operating system. However, mapped file names are only temporary references; name mapping does not produce the same result each time. Therefore, do not build mapped names into your applications. The best course of action is to follow MS-DOS file naming conventions whenever possible. If you follow the suggestions below, file mapping will not be an issue for you or the users of your applications.

- Names can be up to eight characters (without an extension), or up to eleven characters (with a period and three-character extension). Only programs with `.EXE`, `.COM`, or `.BAT` extensions are executable from within MS-DOS.
- MS-DOS names are not case-sensitive, but almost all SunOS commands are lowercase; therefore use lowercase letters for all of your file names.
- Do not use these characters in file names: `" . / [ ] : | < > + = ; ,` You can use a period only as a separator between the file name and an extension.

Issuing SunOS Commands from DOS Windows

The system comes with a number of SunOS commands that you can issue from within DOS Windows, provided that `/etc/dos/unix` is part of the your MS-DOS `path`. These commands are MS-DOS `.COM` programs that point to the actual SunOS commands. They accept the ampersand (&), so you can run them in the background. The preinstalled commands are listed below.

Table 7-1 *Preinstalled SunOS Commands*

```
at
awk
calendar
cat
chgrp
chmod
chown
cmdtool
cmp
comm
cp
csh
cut
date
diff
echo
egrep
fgrep
file
find
grep
head
kill
ln
lpq
lpr
lprm
ls
mail
mailtool
make
man
mesg
mkdir
more
mount
mv
nice
passwd
pr
ps
pwd
rlogin
rm
rmdir
rsh
sed
size
sort
split
stty
tail
tar
tee
time
tr
umount
unix
vi
wc
whatis
whereis
who
write
yppasswd
```

To install additional SunOS commands for your applications, become superuser by entering the su command and then enter:

```
cd /etc/dos/unix
ln -s unix.com newcommand.com
```

at the SunOS prompt.

Text-Only Applications

dos(1) opens a window whenever you invoke most PC programs. However, this is not the case for text-only applications delivered with the Sun386i system. Text-only applications are those that do not attempt to address the cursor, clear the screen, or display graphics. DIR is a good example of such a program. vi is not a text-only application, since it controls the cursor position and makes assumptions about the screen geography.

Text-only applications do not require an 80x25 display. Therefore, if implicit execution is set with the DOSLOOKUP environment variable, dos executes text-only applications in a cmdtool or shelltool window, rather than automatically popping open a new DOS window. You can add to the list of text-only applications that dos recognizes by including the application's name to your setup.pc files, as a value for TEXT. (Refer to page 114 for more information about the setup.pc file.) The list of text-only applications that are shipped with the Sun386i system is shown in Table 7-2.

Table 7-2 *Preinstalled Text-Only Commands*

ATTRIB
ASSIGN
BACKUP
BREAK
CHKDSK
COMMAND
COMP
COPY
DATE
DEBUG
DIR
DISKCOMP
DISKCOPY
EXE2BIN
FDISK
FIND
FORMAT
JOIN
LABEL
LINK
MODE
RECOVER
REPLACE
RESTORE
SELECT
SHARE
SORT
SUBST
SYS
TIME
TREE
TYPE
VER
VERIFY
XCOPY
XDIR

Running make(1) on MS-DOS Targets

In addition, if you specify MS-DOS files as text-only files, you can run compilers, assemblers, and SunOS make(1) files on them in cmdtool or shelltool windows. For example,

```
file.exe: file.c
          dos -w -c cc file.c
```

where *file*.exe is the MS-DOS target file, and *file*.c is the file that the target file depends upon. The -w option to the dos command declares *file*.c as a text-only file, and the -c option indicates that the command, in this case cc, follows.

File Permission Differences

Generally, access to files is the same under SunOS and MS-DOS systems. The exceptions are:

- MS-DOS does not recognize execute restrictions. That is, any user with read permission to a file can execute that file. Without read permission, users cannot execute files.
- Drive C: does not support SunOS file permissions, since the SunOS system cannot directly access files on drive C:. However, because the SunOS system views drive C: as one large file, you can restrict access to all drive C: files to a specific owner or group.

7.3. Peripheral Issues

The MS-DOS drive designations are:

Drives A: and B: — Reserved for diskettes.

Drive C: — A "virtual" hard disk of up to 20 Mbytes, that the SunOS system cannot access; use this drive only to install copy-protected or install-protected PC software.

Drives D: through S: — Virtual hard disks tied to system SunOS directories that can expand as required; use these drives for data files and unprotected PC applications.

All MS-DOS drivers listed in `CONFIG.SYS`, the MS-DOS configuration file, must actually be on drive C:, where `CONFIG.SYS` resides. This is because MS-DOS loads drivers before it begins to communicate with the SunOS system (toward the end of the `AUTOEXEC.BAT` file), and drive C: is the only drive that MS-DOS can access until SunOS communications are activated. The message `Bad or missing` *xxx*`.sys` appears if you try to access a device that has a driver that is not on drive C:.

You can add MS-DOS peripherals either by:

- Adding an AT card that uses an MS-DOS driver provided by the card manufacturer
- Adding an AT card that uses a device-specific driver that you write

Regardless of the method used, you must add information about new drivers to three files—`CONFIG.SYS` (an MS-DOS file), and `setup.pc` and `boards.pc` (two SunOS files described in the following sections). Then you must invoke DOS Windows from the Desktop menu before using the new driver, or enter **dos -s** to save the new driver in `.quickpc`, a quick-start file containing a snap-shot image of MS-DOS. (Refer to *Sun386i Advanced Skills* for more information.)

The `setup.pc` file contains configuration information on all devices attached to a system that users might want to access via MS-DOS, including the SunOS files associated with those devices. The `boards.pc` file contains a list of the boards that only MS-DOS, not the SunOS system, can access on the system. MS-DOS cannot access a peripheral listed in the `setup.pc` file unless it is also in the `boards.pc` file. The `boards.pc` file is in the `/etc/dos/defaults` directory, and `setup.pc` is in the user's home directory, `~/pc/setup.pc`.

setup.pc File

The first time you open DOS Windows, dos creates a pc directory under the home directory, and places a copy of setup.pc there. You can edit this file, but generally should not delete anything in it. A description of the default setup.pc file follows, with number signs (#) indicating comment lines. (Descriptions shown here are not part of the default file.) For more general information about the setup.pc file, refer to *Sun386i Advanced Skills.*

```
# MS-DOS Device          SunOS Device Path Name

A                        /dev/rfd0c
# Diskette device name.

C                        ~/pc/C:
# Drive C: file name.

COM1                     /dev/ttya
# Specifies the serial device attached to the serial port.

LPT1                     lpr
# Specifies how to process MS-DOS LPT1 text; the default is the default printer.

LPT2                     cat >>~/lpt-2
# Specifies how to process MS-DOS LPT2 text; the default is to append to the
# ~/lpt-2 file in the user's home directory.

LPT3                     psfx80 | lpr
# Specifies how to process MS-DOS LPT3 text; the default is Epson™ FX-80
# emulation on the default printer.

SAVE                     ~/pc/.quickpc
# Specifies .quickpc, a quick-start file created with the dos -s command
# that contains a snapshot image of MS-DOS after it has read CONFIG.SYS and
# most of AUTOEXEC.BAT (up to the RUNDOS line). This file is used when any
# dos command other than dos -b (the default command issued when started
# from the Desktop menu) or dos -s is issued. Starting MS-DOS is much
# quicker with the .quickpc file.

# TEXT
# List of user-specified text-only applications, in addition to the standard ones
# shipped with the Sun386i system. Running these applications from the SunOS
# system will send output to the current window instead of opening a new DOS
# window.

# BOARDS
# List of boards from /etc/dos/defaults/boards.pc that you want
# to attempt to access upon opening DOS Windows. If a board is already in use,
# it will appear as detached in the Devices submenu on the Sun386i system. In
# this case, you can release the device from the window that owns it, and then
# attach the device from the current window.
```

`boards.pc` File

As with `setup.pc`, you also can edit the `boards.pc` file; however, unlike `setup.pc`, each system should have only one copy of `boards.pc`, which affects all users on the system. If you add a device to run under MS-DOS, you must include its board name and block I/O information in the file. You must also include interrupt-level information for boards that use interrupt levels, as well as indicate whether or not the device can be shared. The `boards.pc` file included with the Sun386i system contains a list of commonly used boards included as comment lines; you can remove the comment symbols for those boards that you have and want PC applications to use.

The following table shows the AT bus I/O address spaces that `dos` emulates. If you add a card in one of these address spaces, MS-DOS ignores it. If you specify an emulated address in the `boards.pc` file, the next time you open a window the system displays a message stating that the address range is already in use. You can turn off emulation for all but the hard disk by placing a comment character (#) at the start of the pertinent line in `setup.pc`.

Table 7-3 *I/O Address Space Emulation*

Address	*MS-DOS Use*
1F8 – 1FF	Hard disk emulation
230 – 237	Bus mouse emulation
278 – 27F	Parallel port 2
378 – 37F	Parallel port 1
3B0 – 3BF	Monochrome display adapter
3D0 – 3DF	Color display adapter
3F0 – 3F7	Diskette controller

No two boards in the same system can have the same interrupt level. Because many boards have a factory-set interrupt level of 3, occasionally you might have to rejumper the board to set a new interrupt level, as on regular PCs. You must then also change the interrupt-level information in the `boards.pc` file before accessing the attached device. Table 7-4 shows the availability of interrupt levels for the Sun386i system. For more details about adding a board to the `boards.pc` file, refer to *Sun386i Advanced Skills*.

Table 7-4 *Interrupt Level Availability*

Interrupt Level	*Availability*
0	Unavailable; used for timer emulation
1	Unavailable; used for keyboard emulation
2	Unavailable; used for interrupt controller 2 cascade
3	Available for board (specified in `setup.pc`)
4	Available for board, unless COM1 emulation in use (specified in `setup.pc`)
5	Available for board, unless LPT2 emulation in use (specified in `setup.pc`)
6	Unavailable; used for diskette emulation
7	Available for board, unless LPT1 emulation in use (specified in `setup.pc`)
8	Unavailable; used for real-time clock emulation
9	Available for board
10	Available for board
11	Available for board
12	Available for board
13	Unavailable; used for 8087 numeric coprocessor emulation
14	Unavailable; used for hard disk emulation
15	Available for board

7.4. Capabilities and Limitations

This section describes some MS-DOS features and limitations that you should know about. It contains sections on:

- Conversion programs for converting text files from MS-DOS to the SunOS system and vice versa
- Differences between the MS-DOS and SunOS command interpreter
- Determining the DOS Windows number to create unique file names and help avoid network collisions
- 80386 instructions supported
- Limitations such as those relating to screen height, remote port use, certain types of applications, running simultaneous versions of MS-DOS applications, interrupt rates, and space issues

Converting Between MS-DOS and SunOS Text Files

MS-DOS and the SunOS system have slightly different conventions regarding text file delimiters. Consequently, the Sun386i system includes special utilities to convert files from one set of conventions to the other. The program to convert MS-DOS files to SunOS files is called `dos2unix`(1); the program to convert SunOS conventions to MS-DOS conventions is called `unix2dos`(1). The Sun386i system contains two versions of each program, a SunOS version and an MS-DOS version, to make it easier to run both utilities from either system.

Conversion does not happen automatically. You must invoke these programs as necessary, and can do so from either the MS-DOS prompt or the SunOS prompt. Include a source file name and a destination file name on the command line.

You also can use dos2unix and unix2dos in piped expressions. When used in this way, you must explicitly include them in the command line—converting automatically between the two text format conventions would allow binary data to be destroyed automatically. For example, all of the four following commands are legal:

```
dir | dos2unix > filename
dos2unix dos_file > sunos_file
dos2unix dos_file sunos_file
dir | dos2unix | grep ASM
```

Converting Between MS-DOS and ISO Text Files

dos supports the display of 8-bit files containing MS-DOS international characters. However, the MS-DOS character set is different from the ISO (8-bit international ASCII) character set used throughout Europe and supported by the Sun386i system. To display text files containing ISO characters in DOS Windows, you must first convert ISO files with the unix2dos program. Similarly, to display text files containing MS-DOS international characters correctly in a SunOS Text Editor window (the only window that supports 8-bit characters), you must convert files with the dos2unix utility. Cutting and pasting between windows works correctly without any conversion. The Alternative Code Sets section on page 149 provides more information about MS-DOS and ISO text file conversion, and Appendix H contains tables that show how characters are mapped from MS-DOS to ISO and vice versa.

Unexpanded Command Line Interpretation

When you implicitly invoke an MS-DOS command that includes shell variables from the SunOS prompt (implicit invocation is made possible by the environment variable `setenv DOSLOOKUP on` in `.login`), MS-DOS receives the command line in unexpanded form. If, however, you enter an explicit command from the SunOS prompt, such as `dos -c dir *.c`, the shell expands the * before calling dos, which is not the desired effect. Therefore, when explicitly invoking dos, place the shell meta characters in quotes to stipulate that the shell pass the unexpanded command line to MS-DOS. For example, `dos -c "dir *.c"` is the format you should use.

Determining the Window Number

dos allocates a unique window number to each dos process on a Sun386i system. To help ensure that a process running in DOS Windows has a unique name throughout a network, you can incorporate its window number into a file name and store that file in a machine-specific directory such as /tmp. The window number for each dos process is in the byte at f000:42405. This is the same binary number that appears on the namestripe of the window. Numbers are allocated starting at 1, with the lowest available number assigned to each window.

80386 Instructions Supported

This section lists the 80386-specific instructions supported. Be aware, however, that certain MS-DOS compilers and assemblers might not support them. Appendix B contains the 80386 assembly language definition.

The first two columns of Table 7-5 (on the next page) list instructions by name. The third column lists groups of instructions supported; some of these groups represent many individual instructions.

Table 7-5 *80386 Instructions Supported*

bit scan	lfs	byte set on condition
bound	lgs	double-shift instructions
enter	lss	generalized multiply
ins	outs	long displacement conditional jumps
leave	push immediate	move with sign/zero extension
	pusha	single-bit instructions

EDITDOS: Taking Advantage of SunOS and MS-DOS Systems

The EDITDOS program uses textedit and MS-DOS versions of DOS2UNIX and UNIX2DOS to convert and edit MS-DOS text files. EDITDOS:

1. Uses the window number in f000:4205 to ensure a unique file name.
2. Uses DOS2UNIX to copy the file to SunOS format, and onto a SunOS-accessible directory with a unique file name based on the window number.
3. Invokes textedit on the file. Strips the optional ampersand from the command so that the conversion back to MS-DOS format does not occur until the edit is complete. Passes command-line arguments to textedit for changing parameters such as window position and size.
4. Checks the return status from textedit to make sure that the modification occurred without errors.
5. Converts the edited file back to MS-DOS format, and gives the file its original name (placing it in an MS-DOS directory).
6. Cleans temporary files, including illegal MS-DOS file names.

EDITDOS is a good example of how you can use the combined SunOS and MS-DOS environment on the Sun386i system. The next section shows source code for EDITDOS.

EDITDOS Source Code

```
#include <stdio.h>
#include <string.h>
#include <stat.h>
#include <ctype.h>
#include <process.h>
#include <dir.h>
#define TRUE 1
#define FALSE 0

char editname[25]; /* name of work file to be */
                   /* edited in /tmp */
char *dosname;     /* name of DOS file from */
                   /* command line */
char *templine;    /* temporary holding space */
                   /* for spwanlp() commands */
char **args;       /* array for textedit args */
char workname[100];/* unique name for scratch */
                   /* directory */
```

```
struct stat sbuf;
extern char *malloc();
extern char peekb();

main(argc,argv)
int argc;
char **argv;

{
int actr;       /* temporary counter variable */
                /* for argument append loop */
int passctr;    /* temporary counter for trimming
                /* out & symbol */
int spstatus;   /* status returned from */
                /* spawnlp()/spawnvp() call */
int badtext_flag = FALSE; /* if TRUE, skips */
                         /* UNIX2DOS conversion */
int exit_code = 0;
int old_disk;   /* former disk drive letter */

if (argc < 2 )
   {puts("Usage: EDITDOS filename.\n");
    exit(1);
   }
else
  { dosname = malloc(1 + strlen(argv[1]));
    templine = malloc(1000 + strlen(argv[1]));
    strcpy (dosname, argv[1]);/* copy parameter */
                              /* into dos file */
                              /* name */
    if (stat(dosname, &sbuf) != 0) /* file */
           /* doesn't exist or other error */
       { printf("editdos: File %s not found.\n",
         dosname);
         exit(2);
       }

    get_dos_file_name();/* process dosname */
                        /* (global)and put */
                        /* result in editname */
    sprintf(workname,"dos%d", peekb(0xf000,
      0x4205));  /* get DOS window number to */
                 /* create unique file name */

    /* Make temporary directory */
    mkdir("D:\\TMP");
```

```
sprintf (templine,"D:\\TMP\\%s",workname);
if (stat(templine, &sbuf)) /* directory */
                       /* doesn't exist */
   if (mkdir(templine))
      { printf("Could not create D:\\TMP\\%s
          directory.\n",workname);
        printf("File %s not edited.\n",
          dosname);
        exit(3);
      }

/* Convert to SunOS format */
sprintf (templine,"D:\\TMP\\%s\\%s",
  workname, editname);
spstatus = spawnlp(P_WAIT, "DOS2UNIX.EXE",
  "DOS2UNIX", dosname, templine, NULL);

if (spstatus < 0)
   { perror ("Could not run DOS2UNIX. File
       not edited.\n");
     exit_code = 4;
     goto cleanup2;
   }

if (spstatus > 0)
   {
    printf("File %s not edited.\n", dosname);
    exit_code = 4;
    goto cleanup2;
   }
/* Invoke textedit */

args = (char**) malloc ((argc + 5) *
  sizeof(char*));  /* allocate array */

passctr = 0;  /* to track number of args */
args[passctr++] = strdup("textedit");
args[passctr++] = strdup ("-font
 /usr/lib/fonts/fixedwidthfonts/pcfont.r.14");
args[passctr] = malloc (256);
sprintf (args[passctr++],"/tmp/%s/%s",
  workname, editname);

for (actr = 2;actr < argc; actr++)
  /* reappend textedit arguments */
    { if (strcmp (argv[actr], "&") != 0 )
       /* don't allow SunOS background mode */
```

```
             args[passctr++] = strdup(argv[actr]);
        }

    args[passctr] = NULL; /* assign null to */
                    /* signal last argument */

    old_disk = getdisk(); /* assign original */
                          /* path to old_disk */
    setdisk('D'-'A');  /* set to drive D to */
                       /* invoke textedit */
    spstatus = spawnvp(P_WAIT, "TEXTEDIT.COM",
      args);
    setdisk(old_disk); /* restore to original */
                       /* path */

    if (spstatus < 0)
       { perror ("Could not run textedit. File
         not edited.\n");
         exit_code = 5;
         goto cleanup;
       }

    if (spstatus > 0)
       { printf("Error editing /tmp/%s/%s in
         textedit.\n",workname, editname);
         printf("File not edited.\n");
         exit_code = 5;
         goto cleanup;
       }
   /* Convert back to DOS format if edit was OK */

sprintf(templine,"D:\\TMP\\%s\\%s",workname,
  editname);
spstatus = spawnlp(P_WAIT, "UNIX2DOS.EXE",
  "UNIX2DOS", templine, dosname, NULL);

if (spstatus < 0)
   { perror ("Could not run UNIX2DOS.\n");
     exit_code = 6;
     goto cleanup;
   }

if (spstatus > 0)
   { printf("Problem converting from SunOS to
       DOS.\n");
     printf("Backup work file is stored in
       /tmp/%s/%s.\n",workname, editname);
     exit_code = 6;
     goto cleanup;
   }

cleanup: ;

/* delete temporary directory, temporary files */
setdisk('D'-'A');
sprintf(templine,"rm -f /tmp/%s/%s{,%,%%}",
  workname, editname);
spawnlp(P_WAIT, "UNIX.COM", "unix", templine,
  (char *)NULL);
sprintf(templine,"/tmp/%s", workname);
rmdir(templine);
setdisk(old_disk);

cleanup2: ;

   }
exit (exit_code);
}

get_dos_file_name()
{
int ln,
    ctr,
    foundflag = 0,
    startpoint = 0,
    eln;

ln = strlen(dosname);

for (ctr = 0;ctr <= ln; ctr++)  /* flip slash */
                                /* if needed */
    { if (dosname[ctr] == '/')
          dosname[ctr] = '\\';
    }

for (ctr = ln - 1;ctr >= 0; ctr --)
    { if ((dosname[ctr] == '\\') ||
       /* hit first delimiter in DOS file */
          (dosname[ctr] == ':'))
             /* name (reading backwards) */
          { startpoint = ctr + 1;
            foundflag = TRUE;
            break;
          }
    }
if (foundflag)
   strcpy(editname, &dosname[startpoint]);
     /* copy last portion of DOS name */
else
   strcpy(editname, dosname);   /* copy entire */
                                /* name */


eln = strlen(editname);
for (ctr = 0; ctr <= eln; ctr ++)  /* convert to */
                                   /* lower case */
     {
     editname[ctr] = tolower(editname[ctr]);
     }
}
```

MS-DOS Limitations

While MS-DOS offers many benefits, most importantly the ability to run PC applications on the Sun386i system, it does have some limitations, listed below.

Screen Height

Depending on the screen height, it may not be possible to maintain two full DOS Windows (for multiple, simultaneous PC applications) on the screen at one time without overlapping.

Port Allocation and Access

You cannot allocate or access serial ports, parallel ports, diskette drives, or AT bus ports on either remote systems or on PCs attached through PC-NFS. This limitation applies to both PC and SunOS applications.

Protected-mode Instructions

The Sun386i system does not support 80286 protected-mode opcodes, which are used by several system software products developed for the IBM PC/AT®. Protected-mode instructions follow.

Table 7-6 *Protected-Mode Instructions*

- `arpl`
- `clts`
- `hlt`
- `lar`
- `lgdt`
- `lidt`
- `lldt`
- `lmsw`
- `lsl`
- `ltr`
- `mov` (to/from debug registers, test registers, control registers)
- `sgdt`
- `sidt`
- `sldt`
- `smsw`
- `str`
- `verr`
- `verw`

Interrupt Rates

The Sun386i system can't service interrupts at the same rate as on a PC. This results in some boards not operating due to lack of interrupt service.

Data Acquisition Applications

Data acquisition applications are generally not well-suited to DOS Windows, because they rely on uninterrupted access to devices, as well as dedicated processor time to receive data. For this type of application, a smart card, capable of storing data in buffers, works better.

Simultaneous Running of Some Applications

Unless a PC application specifically states that it is for use in a multiuser environment, don't run the same application simultaneously in different DOS Windows. This is because some applications, such as certain word processors and database packages, use scratch files for temporary storage. You could inadvertently save changes from multiple invocations of an application in the same scratch file.

Drive C: Space

Drive C: initially contains only the File Allocation Table (FAT), root directory, and a few standard MS-DOS files. As files are added, the SunOS file containing drive C: will expand. There will not be any way to reclaim the space held by deleted files on drive C:, other than copying the drive C: files to another drive, deleting the drive C: file, and recreating it. The maximum size of the drive C: file is 20 Mbytes.

Development Environment

As a development environment, the SunOS system on the Sun386i is far superior to MS-DOS on a PC. You might want to consider not only porting PC applications to the Sun386i system, but converting them to run under the SunOS system, which runs faster. In addition, the SunOS system provides:

- Larger process address space
- Flexible utility programs, compilers, and debuggers
- Powerful system calls and libraries
- 32-bit CPU performance and enhanced instruction set
- More sophisticated graphics capabilities

7.5. Communication Between Commands and Applications

It is possible to pipe output from MS-DOS commands to SunOS commands and vice versa. This means that you can mix SunOS commands with MS-DOS commands and can pipe the output from one program to the input of another program, in accordance with the normal rules and syntax for piping. Similarly, you can pipe into and between PC applications. This section also describes how you can use named pipes to communicate between SunOS and MS-DOS processes. In addition, you can include SunOS commands in MS-DOS batch files, and can run the files from the MS-DOS prompt. SunOS commands in files must follow the rules for entering them at the shell prompt. You can also copy and paste between systems, and can share text or binary files.

Invoking MS-DOS Commands at the SunOS Prompt

When a command is entered at a SunOS shell prompt, the system searches for the command according to the SunOS shell's traditional search path. SunOS commands, when found, are executed normally. If the requested command is not found along the SunOS path, the shell attempts to execute an implicit `dos` process, using the version of MS-DOS specified by the `.quickpc` file; the `setenv DOSLOOKUP on` variable must also be included in the `.login` file for implicit invocation.

MS-DOS then attempts to locate the requested program, using the MS-DOS path to determine directory search order. If MS-DOS finds the program (on either the current drive, the C: drive, or the requested path), it returns a success indication to the SunOS shell upon termination. When the command completes, the `Press any key to continue` message is displayed. If it cannot locate the requested program (for instance, if the program name is misspelled), the shell displays the usual `Command not found` message. This allows MS-DOS and the SunOS system to avoid path name differences and makes the search order for programs completely unambiguous.

For applications that dominate the screen (all but text-only applications), the system automatically invokes a new DOS window. When this window appears, the SunOS directory from the previously active SunOS shell becomes the current MS-DOS directory. This automatically generated window remains displayed as long as the PC program that opened the window is running. When the PC application terminates, the window disappears and the exit code from MS-DOS is returned to the SunOS system. To view the screen contents after the PC process has terminated (for a full-screen application), explicitly invoke DOS Windows either from the menu or by entering **dos** from the SunOS command line.

Invoking SunOS Commands at the MS-DOS Prompt

SunOS commands are links installed in the `/etc/dos/unix` directory. This directory must be part of the MS-DOS `path` to allow invocation of SunOS commands from within DOS Windows. When you enter a SunOS command from DOS Windows, a new window is displayed for command execution. When the command completes, the `Press any key to return to DOS` message is displayed. The exit code from the SunOS program is returned to MS-DOS to enable deletion of status information. If the environment variable `setenv DOS_CMDTOOL on` is in your `.login` file, the results of SunOS commands entered in DOS Windows are displayed in a `cmdtool` window, instead of in DOS Windows.

Piping Between Commands and Between Applications

You can pipe information between MS-DOS and SunOS commands, provided you initiate the MS-DOS command from a `cmdtool` window (not from DOS Windows). For example, `dir | grep 87` shows all files on drive C: that were created or last updated in 1987. If `DOSLOOKUP` is not set in your `.login` file, use the command `dos -c dir | grep 87` instead.

Similarly, you can pipe information into any PC application that you started in a `cmdtool` window. However, some SunOS applications designed to run exclusively under SunView (generally those making extensive use of the mouse) usually don't accept entry from standard input; therefore, you can't pipe information *into* some SunOS applications. In addition, many PC applications don't send their output to standard out; consequently, these programs might not work correctly with pipes.

Using Named Pipes

You can also use named pipes to establish communication between PC and SunOS applications by using the mknod(2, 8) command with the p option, as shown by the following example:

```
(at SunOS prompt):  /etc/mknod pipe p
(at DOS prompt):  tree >> pipe
(at SunOS prompt):  more pipe
```

Here's an example showing the other direction:

```
(at SunOS prompt):  /etc/mknod pipe p
(at DOS prompt):  /files/loaded/appl/games/games/
   fortune > pipe
(at DOS prompt):  unix2dos < pipe
```

Be careful when using named pipes. For instance, make sure that the pipe is a legal MS-DOS file name (refer to Naming Your PC Applications on page 111). Also, from MS-DOS, always append to the pipe. If you don't, MS-DOS tries to close the pipe before writing to it, and the reader receives an end-of-file notification. Pipe readers and writers block when the pipe is empty (readers) or full (writers).

Background Mode Considerations

The ampersand (&) character at the end of a command line tells the SunOS system to perform the operation in background mode, freeing you to continue with other work. While you might want to take advantage of this feature, there are also times when you should avoid using it. For instance, don't use & if you want to synchronize completion of the SunOS command with your MS-DOS program or batch file. Another example follows:

```
make >& errors & unix2dos errors
```

The make(1) command will return quickly, since it's running in the background, but the error log file will not be ready for viewing until the make command finishes.

8

Peripheral Devices

8

Peripheral Devices

This chapter provides a brief overview of peripheral devices on the Sun386i system. It discusses how users can add devices and includes information about MS-DOS and SunOS device drivers, as well as about the SCSI interface. For detailed information required to write device drivers for your own applications, refer to *Writing Device Drivers for the Sun Workstation.*

8.1. Adding Devices

You can add a device to the Sun386i system by:

- Installing a card in one of the three AT slots or one XT slot provided in the system enclosure. Cards in these slots communicate with the main processor via the AT bus.
- Connecting the device to the SCSI interface. SCSI connectors are provided on the back of both the system unit and expansion unit. Peripherals in the expansion unit are themselves connected to the SCSI interface (see System Interfaces and Mass Storage on page 19 and the Expansion Unit section on page 160 of Appendix A).

You can use AT cards to add devices to run under either the SunOS or MS-DOS operating system. If you use an AT card to add a device, you must write a device-specific driver and integrate it into the operating system. Using the SCSI interface you can add 327 Mbyte formatted SCSI Winchester disks or SCSI compatible/Sun-compatible streamer tape devices supplied by Sun. In addition, if you have a SunOS system source license, you can write your own SCSI driver and integrate it into the SunOS system.

Integration is considerably easier with the SunOS 4.0 system on the Sun386i workstation because you can write drivers that can be dynamically loaded into the kernel at any time. Drivers that are not linked into the kernel are called *loadable drivers.* If you write a loadable driver, you don't have to rebuild and reboot the kernel to add the driver to the system. After writing the driver, simply use the `modload`(8) command to load the driver into a running system. You also can convert existing drivers to loadable drivers. *Writing Device Drivers for the Sun Workstation* provides additional information and examples.

MS-DOS Drivers

Users can add AT cards that use MS-DOS drivers provided by the card manufacturer without modifying the drivers. That is, anyone can load and run MS-DOS cards and drivers from DOS Windows (see Chapter 7) in the same manner as on a PC. Users can only access devices attached to MS-DOS drivers from within MS-DOS. Because MS-DOS drivers come already compiled and linked, a user merely:

1. Shuts off the system
2. Adds the plug-in board to the system
3. Powers the system back on and reboots the SunOS system
4. Starts the `dos` program
5. Inserts the installation diskette that accompanied the board into Drive A:
6. Adds the device via DOS Windows and modifies the `CONFIG.SYS` file
7. Exits from DOS Windows
8. Modifies the `boards.pc` and `setup.pc` files to include information about the new driver
9. Enters the `dos -s` command and then opens DOS Windows

If users add a card that has the same interrupt request line as a card already on their system, then they also must rejumper the card (according to the supplier's instructions) and include the new information in the `boards.pc` file.

MS-DOS drivers run under the control of the SunOS system and DOS Windows, but the drivers are unaware of this. Because the SunOS system could switch control to another task during device operation, strict timing dependencies might fail. If a peripheral and controller have strict timing requirements, you should write its driver as a SunOS driver. Sun does not provide any documentation about how to write MS-DOS drivers. If you must write a driver to run specifically under MS-DOS, refer to the *IBM DOS Technical Reference* or *Advanced MS-DOS* by Ray Duncan (Microsoft Press).

SunOS Drivers

Unlike MS-DOS drivers, which can provide services only to MS-DOS programs, SunOS drivers can provide services to both SunOS and MS-DOS programs.

The drivers for devices intended to run directly under the SunOS system are similar to other SunOS drivers with which you may be familiar. Four new interface routines exist to let you obtain services for Sun386i system devices; they are briefly described below. The first two routines are for devices that do DMA transfers and the third and fourth ones are for devices that use the I/O space ports.

- `dma_setup` — sets up all of the information required to prepare the DMA channel on the Sun386i system
- `dma_done` — frees the DMA channel (82380) after a DMA transfer is completed, enabling another transfer to occur
- `outb` — sends a byte value to the I/O address specified (many Sun386i system devices, such as diskette drives, can only be accessed by using I/O addresses)
- `inb` — reads and returns the byte value from the specified port address in the I/O space

These new routines, as well as those that you might have already used, are documented in *Writing Device Drivers for the Sun Workstation.* This same manual also contains a sample parallel port driver for the Sun386i system. If you're writing your own drivers, you should also refer to the kadb(8S) man page in the *SunOS Reference Manual* for a description of debugger options.

8.2. AT Bus Description and Issues

The AT bus interface couples the 32-bit 80386-based Sun386i system to an AT bus structure. This enables PC applications to run on the Sun386i system. This section includes information about

- AT bus operation
- Memory-mapped I/O
- Interrupt channels
- DMA channels
- Bus signals
- Limitations

AT Bus Operation

The functional input of the interface logic is a 32-bit address/data bus. The output is an 8- or 16-bit data and 20- or 24-bit address AT bus. The hardware automatically converts the 32-bit data bus to the 8/16-bit data bus with multiple (if necessary) AT bus cycles. The 80386 processor is momentarily wait-stated until the transaction is completed.

The 32-bit address bus is buffered and the low-order 24 bits are driven across the AT bus. The byte/word address (A0, A1, and BHE) required by the AT bus is based on the byte-select lines of the 80386 processor. This means that the AT bus address space must be on an even 16 Mbyte boundary.

The AT bus interface logic connects the 7 AT bus DMA channels to 7 system DMA channels. The AT DMA controller's I/O addresses are under software emulation for MS-DOS applications. Similarly, the interface logic connects the 11 AT bus interrupt levels to 11 system interrupt controller levels. Again, the interrupt controller's I/O addresses are under software emulation for MS-DOS applications.

Finally, the hardware has selectable timings to compensate for (16-25 MHz) base system clocks. This ensures that the AT bus timings are marginally better than those of the original IBM AT. Also, synchronizing the timing logic (state machine) with the main processor eliminates any asynchronous problems (such as metastable conditions) between them.

Memory-Mapped I/O

Some important points about AT bus memory on the Sun386i system are listed below.

Mapping — The Intel 80386 processor handles both memory mapping and I/O mapping; memory-mapped I/O provides additional programming flexibility.

I/O port access — Any memory instruction can access any I/O port located in the memory space (MOV transfers data between any register and any port; AND, OR, and TEST manipulate bits in the internal registers of a device).

Reading registers — On some devices, reading a register will not read back what was written. Therefore, instructions such as AND, OR, and TEST can sometimes produce unexpected results.

Memory-mapped I/O — Memory-mapped I/O performed with the full instruction set maintains the full complement of addressing modes for selecting the desired I/O device. The 16 Mbytes of AT memory is mapped into the 4 gigabyte address space of the Sun386i system at `0xE000 0000`. Within this 16 Mbytes, physical addresses are mapped one-to-one with virtual addresses.

IN, OUT, INS, OUTS instructions — All I/O transfers using the `IN`, `OUT`, `INS`, and `OUTS` instructions are performed via the `AL` (8-bit), `AX` (16-bit), or `EAX` (32-bit) registers. The entire 64 Kbyte I/O space is indirectly addressable through the `DX` register.

Interrupt Channels

The Sun386i system has 21 interrupt channels, 11 of which are available to the AT bus. Of these, two are used for the diskette drive and parallel port. The remaining nine channels you can use for AT cards are shown below. Each AT card must have its own interrupt channel; you cannot use the same channel for two cards.

Table 8-1 *Interrupt Channel Assignments*

Interrupt Channel	*Assigned To*
3	AT pin B25
4	AT pin B24
5	AT pin B23
7	AT pin B21
9	AT pin B04
10	AT pin D03
11	AT pin D04
12	AT pin D05
15	AT pin D06

DMA Channels

Some AT cards also use DMA. Table 8-1 shows the Sun386i system DMA channel assignments and sizes. No two AT cards can use the same DMA channel.

Table 8-2 *DMA Channel Assignments*

DMA Channel	*Assigned To*	*Size (bits)*
0	AT bus	16
1	AT bus	8
2	AT bus	8
3	AT bus	8
4	Software	Not available
5	AT bus	16
6	Ethernet	Not available
7	SCSI	Not available

AT Bus Signals

The following table describes the AT bus signals. All signals are TTL-compatible. Direction comments are relative to the main system; for example, outputs are signals driven by the CPU and accepted by the AT bus interface. Each signal is described in detail in the notes following Table 8-2.

Table 8-3 *AT Bus Signals*

Signal Name	*Description*	*Type*
SA<19:0>	System Address	Output
LA<23:17>	Lurp Address	Output
SD<15:0>	System Data	I/O
IRQ<15:14> IRQ<12:9> IRQ<7:3>	Interrupt Request	Input
DRQ<7:5> DRQ<3:0>	DMA Request	Input
-DACK<7:5> -DACK<3:0>	DMA Acknowledge	Output
-IOCHCK	I/O Channel Check	Input
-RESET	System Reset	Output
AEN	Address Enable	Output
-REFRESH	System Refresh	Output
-MASTER	System Master	Input
TC	Terminal Count	Output
SBHE	Byte High Enable	Output
-IOR	I/O Read	Output
-IOW	I/O Write	Output
-SMEMR	Memory Read (lower 1 Mbyte)	Output
-SMEMW	Memory Write (lower 1 Mbyte)	Output
-MEMR	Memory Read (full 16 Mbyte)	I/O
-MEMW	Memory Write (full 16 Mbyte)	I/O
CLK	Synchronous Bus Clock (6 - 8 MHz)	Output
BALE	Address Latch Enable	Output
IOCHRDY	AT Channel Ready	Input
-MEMCS16	Memory 16 chip select	Input
-IOCS16	I/O 16 chip select	Input
OSC	Oscillator (14.31818 MHz)	Output
OWS	Zero Wait State (2 AT clock cycle)	Output

SA0 to SA19 (OUTPUT, Master 3-state)

System Address bits 0 through 19 are used to address memory and I/O devices within the AT bus system. SA0 to SA19 are gated on the AT bus when BALE is high; they are latched on the falling edge of BALE. These address lines could be driven by an AT bus master.

LA17 to LA23 (OUTPUT, Master 3-state)

These signals are used to address memory and I/O devices within the system. These additional address lines give up to 16 Mbytes of addressibility. They have the same timing relationships as SA0 to SA19. These address lines could be driven by an AT bus master .

SD0 to SD15 (I/O)

These signals provide data bus bits 0 through 15 for the memory and I/O devices. D0 is the least-significant bit. All 8-bit devices on the bus should use D0 through D7 for communications to the system microprocessor. To support 8-bit devices, the data from each of the four bytes (32 bits) of the microprocessor are, in turn, gated to D0 through D7 during 8-bit transfers to these devices. Similarly, 16-bit devices are supported by gating the appropriate byte/word during transfers to these devices. 16-bit transfers to 8-bit devices are converted to two 8-bit transfers, and 32-bit transfers to 16-bit devices are converted to two 16-bit transfers, with support for everything in between.

IRQ3 to IRQ7, IRQ9 to IRQ12 and IRQ14 to IRQ15 (INPUT)

Interrupt Requests 3 through 7, 9 through 12, and 14 through 15 are used to signal the microprocessor that an I/O device needs attention. These interrupt requests are redirected to a programmed interrupt request level of the system interrupt controller. An interrupt request is generated when an IRQ line is raised from low to high. The line must be held high until the microprocessor acknowledges the interrupt request (Interrupt Service routine).

DRQ0 to DRQ3 and DRQ5 to DRQ7 (INPUT)

Direct Memory Access 0 through 3 and 5 through 7 are asynchronous channel requests used by peripheral devices and the I/O channel microprocessors to gain DMA service (or control the AT bus). Since the DMA controller can transfer data in any size format, the DMA channels are not limited by the width of the transfer. The DMA controller's control registers are under software emulation and can create the same characteristics as the AT DMA channels.

–DACK0 to –DACK3 and –DACK5 to –DACK7 (OUTPUT)

Direct Memory Access Acknowledge 0 to 3 and 5 to 7 are used to acknowledge the corresponding DMA requests (DRQ0 through DRQ7). They are active low.

–IOCHCK (INPUT)

I/O Channel Check, if enabled, causes an NMI to the system processor. This signal is active low.

–RESET (OUTPUT)

This signal resets or initializes channel devices at power-up or when commanded by the microprocessor.

AEN (OUTPUT)

Address Enable indicates when the DMA controller has control of the address bus, data bus, and controls. This is the same as the HOLD ACKNOWLEDGE line between the microprocessor and the DMA controller. It is driven low if an AT bus master controls the current cycle.

–REFRESH (OUTPUT, Master 3-state)

This signal indicates a refresh cycle is taking place. It can be driven by the bus master only for AT bus RAM.

–MASTER (INPUT)

The master can issue a DRQ; when it receives a –DACK, it issues a –MASTER which gives it sole access to the AT bus.

TC (OUTPUT)

Terminal Count provides a pulse when the terminal count for any selected DMA channel is reached.

SBHE (OUTPUT, Master 3-state)

Byte High Enable indicates when SD8 through SD15 are valid. 16-bit devices use BHE to condition data bus buffers tied to SD8 – SD15.

–IOR (OUTPUT)

This signal instructs an I/O device that an I/O instruction data read cycle is taking place.

-IOW (OUTPUT)
This signal informs an I/O device that an I/O instruction data write cycle is taking place.

-SMEMR (OUTPUT)
The hardware decodes the immediate address on the LA20 to LA23 lines for 0000 (lower 1 Mbyte addresses), then ANDs the active result with active -MEMR to form the active -S MEMR signal. (AT bus masters can take advantage of this logic.)

-SMEMW (OUTPUT)
The hardware decodes the immediate address on the LA20 to LA23 lines for 0000 (lower 1 Mbyte addresses), then ANDs the active result with active -MEMW to generate the active -SMEMW signal. (AT bus masters can take advantage of this logic.)

-MEMR (I/O)
As an output, this signal defines a memory read cycle. As an input, this signal is functionally ANDed with LA20 to LA23 = 0000 (lower 1 Mbyte) with -S MEMR as the result.

-MEMW (I/O)
As an output, this signal defines a memory write cycle. As an input, this signal is functionally ANDed with LA20 to LA23 = 0000 (lower 1 Mbyte) with -S MEMW as the result.

CLK (OUTPUT)
This is a 6 to 8 MHz clock that is synchronous with the main 80386 clock. The CLK has a 50% duty cycle. This signal is normally used only for synchronization of signals to the AT bus control signals.

BALE (OUTPUT)
Address Latch Enable defines when there is a valid address on the AT bus. This signal is driven high during all DMA cycles and REFRESH cycles.

IOCHRDY (INPUT)
I/O Channel Ready is pulled low by a memory or I/O device to lengthen the current cycle. This signal is open collector and should have a maximum wait state of no longer than 2.5 microseconds.

-MEM CS 16 (INPUT)
This notifies the control logic that the current memory data cycle is a 16-bit transfer.

-IO CS 16 (INPUT)
This notifies the control logic that the current I/O data cycle is a 16-bit transfer.

OSC (OUTPUT)
This oscillator runs at 14.31818 MHz. It is free running and not tied to the main board in any way.

OWS (OUTPUT)
This notifies the control logic that a two AT bus clock cycle is in a zero wait state for the current cycle.

Limitation

One limitation exists. The AT bus does not support cards operating in "master mode." This means that AT cards do not have direct access to system memory on the System bus. This is not particularly serious because:

- Few cards attempt to operate in master mode
- Direct access is available to memory cards located on the AT bus itself

9

Applications Delivery

9

Applications Delivery

This chapter describes software delivery — both how Sun delivers its software for the Sun386i system and the preferred method for you to deliver your applications for this system. The first part of the chapter discusses the division and distribution of system software in two major parts, and the groups of files, called *clusters*, constituting those parts. The last section describes the steps you should follow to enable users to easily install your applications.

9.1. System Software Overview

Sun386i system software is divided into two major sections: Application SunOS and SunOS Developer's Toolkit. Figure 9-1 below shows the two major divisions of system software and their subsets.

Application SunOS (unbundled)

- Hardware Diagnostics (separate diskette)
- Core System (shipped on disk)
- Optional Clusters (on diskettes or tape)
- Recovery Software (on diskettes or tape)

SunOS Developer's Toolkit (unbundled; loadable clusters on diskettes or tape)

Figure 9-1 *System Software Divisions*

Each site receives the core system on the Sun386i system disk, as well as the *Sun386i Owner's Set* documentation. All other system software and documentation is unbundled; that is, users must purchase both Application SunOS and Developer's Toolkit to have full SunOS 4.0 functionality and accompanying documentation. Sun strongly recommends that each user site purchase at least one copy of Application SunOS. By doing so, a site receives all four pieces of Application SunOS shown in Figure 9-1, plus the *Owner's Supplement Documentation Set.* The next section describes Application SunOS more fully. In addition, users can order the Developer's Toolkit, discussed in Section 9.3 on page 143.

9.2. Application SunOS

Application SunOS includes:

- Hardware Diagnostics — a set of standalone diagnostics available on diskette
- Core system — the base system, providing the ability to run most commercially available Sun and third-party applications; shipped on the Sun386i disk
- Optional clusters — additional software for capabilities such as extended mail and extended networking; available on diskettes or tape
- Recovery software — a backup version of the core system, available on diskettes or tape

Users can purchase Application SunOS on diskettes (approximately 26) or quarter-inch tape (two). In addition to the software listed above, users who purchase Application SunOS also receive the *Sun386i Owner's Supplement Documentation Set.* The following subsections provide details of each Application SunOS subset.

Hardware Diagnostics

The first part of Application SunOS is a set of standalone Hardware Diagnostics. These programs do not require the SunOS operating system. You should run hardware diagnostics when:

- You cannot start your system
- The system displays numerous messages indicating a hardware problem
- You have upgraded your system with a new frame buffer, memory board, or hard disk (to make sure that the new part works properly)
- Your system crashes repeatedly

For information about how to run Hardware Diagnostics and the individual tests they perform, refer to *Sun386i System Setup and Maintenance.*

Core System

The core system provides the minimum subset of the SunOS operating system required by every user. It is sufficient to allow users to operate most commercially available Sun and third-party applications.

The core system includes software that users always need. It uses about 19 Mbytes of disk space and is preloaded by Sun manufacturing on the formatted hard disk that comes with each system (either 91 or 327 Mbytes). There is no automated method to remove any part of the core system, since users should leave all of it on the disk.

The core system includes the groups of files listed below. The file `/usr/lib/load/filesizes` contains the names and sizes of files in each group.

base_root — the root directory (`/`), which includes the kernel; system databases and start-up files; single-user mode requirements; commands such as `automount`(8), `chown`(8), `fastboot`(8), `fasthalt`(8), and `reboot`(8); the `adb`(1) and `kadb`(8S) debuggers; and the file `/usr/lib/load/filesizes`

sunview — SunView tools, icons, commands, and demos, as well as all Sun screen fonts

boot_server — the boot server for booting diskless Sun386i systems from Sun386i system servers, as well as the boot server enabling the Sun386i system to be a server for Sun-2, Sun-3, and Sun-4 systems

encryption — file encryption commands such as `des`(1)

calendar — `calendar`(1) program and required files

basic_commands — most commonly used user commands such as `date`(1V), `grep`(1V), `arch`(1), `csh`(1), `passwd`(1), `crontab`(1), and `kill`(1), as well as `ed`(1), `ex`(1), and `vi`(1) editors

mail — basic mail directories, files, and commands such as `mail`(1), `biff`(1), `sendmail`(8), and `newaliases`(8)

at_commands — commands such as `at`(1), `atq`(1), and `atrm`(1), for executing commands or scripts at a later time

print_spooler — printing commands such as `lpc`(8), `lpd`(8), `lpq`(1), `lpr`(1), and `lprm`(1)

non_readonly — configuration files, spool directories, and other nonread-only files required by optional software such as the Network File System (NFS), the print spooler, extended mail, the audit trail maintenance package, and `uucp`(1C) and `tip`(1C)

sun_specific_commands — commands such as `click`(1) and `screenblank`(1)

online_help — Spot Help and Help Viewer files, plus new kernel error messages

key — encryption keys for secure networking; includes `chkey`(1), `keylogin`(1), `keyserv`(8C), and `keyenvoy`(8C)

basic_networking, rpc_base, and **nfs** — contain networking software, with the exception of the boot server; includes network configuration files, daemons, and administrative and user commands such as `ping`(8C), `rmt`(8C), `rcp`(1C), and `rlogin`(1C)

ease_of_use — `snap`(1) and `organizer`(1) programs

yellow_pages — the Yellow Pages database

dos — MS-DOS 3.3, required to run DOS Windows

load — the commands required to load and unload clusters, including `load`(1), `unload`(1), `loadc`(1), `unloadc`(1), and `cluster`(1)

Optional Clusters

The optional clusters included with Application SunOS are comprised of sets of related programs and files that users might need on the hard disk, in addition to the core system. Users can add individual clusters after installation by using the load(1) or loadc(1) commands (or the snap(1) administration tool), and can subsequently remove them by using the unload(1) and unloadc(1) commands. (Pages 12-13 describe all four commands.) When all optional clusters are loaded, they take about 14 Mbytes of disk space. A file that's part of the core system, /usr/lib/load/filesizes, lists the sizes of these and all other system software files. *Sun386i System Setup and Maintenance* provides more details about the contents of these clusters, listed below.

mail_plus — extended mail commands such as from(1), vacation(1), uuencode(1C), uudecode(1C), and mailstats(8)

spellcheck — spell(1) program and related commands

accounting — basic accounting programs and commands such as ac(8), accton(8), pac(8), last(1), and lastcomm(1)

sysV_commands — basic System V commands such as uname(1V), echo(1V), expr(1V), cat(1V), grep(1V), sdiff(1), and chmod(1V)

advanced_admin — advanced system administration commands such as chroot(8), dump(8), and restore(8)

extended_commands — additional commands such as pagesize(1), trace(1), logger(1), and script(1)

networking_plus — extended networking utilities and commands such as in.fingerd, in.ftpd, and in.rwhod daemons and finger(1), ftp(1C), rwho(1C), gettable(8C), and rpcinfo(8C) commands

audit — audit trail maintenance package, including the auditd and rpc.pwdauthd daemons and audit(8), audit_warn(8), praudit(8), and C2conv(8)

comm — uucp(1C), tip(1C), and related commands

doc_prep — text processing tools such as nroff(1), troff(1), neqn(1), and tbl(1), and directories required to run them

disk_quotas — quota commands such as quot(8), edquota(8), and quotacheck(8)

name_server — in.named, in.tnamed, and sendmail.mx daemons

man_pages — on-line man pages and man commands

plot — plotting commands such as plot(1G) and spline(1G)

old_commands — backward-compatible commands such as make(1), perfmon(1), clocktool(1), setkeys(1), and syslog(1)

games — on-line games, including backgammon, Boggle, and cribbage

Recovery Software

Application SunOS includes recovery software for reloading the core system, if necessary. Recovery software is available on tape and diskettes. *Sun386i System Setup and Maintenance* describes how to load this software, should you need it.

9.3. SunOS Developer's Toolkit

SunOS Developer's Toolkit is a complement to Application SunOS, not a superset. It provides everything missing in Application SunOS needed to achieve full SunOS 4.0 functionality. In addition, the *Sun386i Developer's Toolkit Documentation Set* accompanies each copy of the Developer's Toolkit purchased. The Developer's Toolkit is available on diskettes or quarter-inch tape. As with Application SunOS, you can add and remove Developer's Toolkit clusters individually with the `load`(1) and `unload`(1) commands described briefly in the next section. The file `/usr/lib/load/filesizes`, part of the core system, lists the sizes of these and all other system software files.

SunOS Developer's Toolkit includes the software listed below.

base_devel — software development commands and utilities such as the C compiler, assembler, link editor, `dbx`(1); you must load this cluster to be able to use any of the Developer's Toolkit with the exception of the `help_guide` cluster, which does not require `base_devel`

config — System V files necessary to reconfigure the kernel such as `config`(8) and the `/usr/sys` directory

sunview_devel — SunView development libraries required for writing window-based applications

help_guide — *Help Writer's Handbook* for writing on-screen help for applications (the sections Spot Help Interface and Help Viewer Interface on pages 80–94 contain much of the same information)

plot_devel — libraries such as `libplot.a` and `libplotbg.a` for development plotting functions

proflibs — profiled libraries (denoted by the suffix `_p.a`) such as `libc_p.a`, `libm_p.a`, and `libcurses_p.a`

sccs — commands required by SCCS, the Source Code Control System

sysV_devel — libraries required to port System V applications, including utilities in `/usr/5bin` and `/usr/5lib` directories

9.4. Loading and Unloading Clusters

You can issue the `load`(1), `loadc`(1), `unload`(1), `unloadc`(1), and `cluster`(1) commands to:

- Add one or more Application SunOS or Developer's Toolkit clusters to the disk after installation
- Remove one or more clusters to make space for additional ones
- Display the name of a cluster containing a specified file
- Display a summary of all Application SunOS and Developer's Toolkit clusters, including a cluster's size and whether or not it is loaded

Pages 12-13 provide more information about these commands.

9.5. Releasing Your Software

Users can install your software on the Sun386i system by selecting a few panel items if you follow the steps in this section. These steps also could save you time since they should simplify and shorten your installation instructions.

The preferred method of releasing software on tape or diskette consists of three parts:

1. Creating a file containing copyright information and a brief description of your application.
2. Creating an installation script.
3. Using the `bar`(1) command (available on the Sun386i system only) to place the two files mentioned above and your application onto a formatted (`fdformat`(8)) diskette or tape, or using the `tar`(1) command to place them onto tape for workstations other than Sun386i systems.

Copyright and Description File

The copyright and description file is an ASCII text file that users will read before installing your application with `snap`(1). You must name this file `copyright` because this is the file name that `snap` looks for. There is no restriction on the size of either file, but users will only see 15 lines of the `copyright` file.

Installation Script

Installation scripts will vary between applications in content and complexity. The script must be named `Installscript`, and the basic things it should do are shown below.

1. Make sure that there's enough disk space for the application in `/usr/local`. (On Sun386i systems, `/usr/local` is a link that resolves to `/files`<*n*>`/local`.) If there isn't, display a message in a Commands window stating this, and instructing an administrator to enter a different location.
2. Make `/usr/local` or the directory entered by an administrator the working directory (using `cd` *directory_name*).
3. Check to see if *application_name* exists. If it doesn't, create an *application_name* directory (with `mkdir`(1)) and beneath that create <*arch*>.<*OS release*>, `share`, and `language` subdirectories, as described on pages 206–207. <*arch*> can be either `sun1`, `sun2`, `sun3`, `sun4`, or `sun386` and <*OS release*> has the format `SunOS4.0`, `SunOS4.1BETA1`, `SunOS4.1BETA2`, and so on.
4. Extract the application files from the tape or diskette and place them into the directories just created.

If you are including icons for your application's files or on-screen help for your application, see pages 68 and 96 in Chapter 6 for specifics about what your installation script should contain.

NOTE *Do not program any absolute path names into your application. Regardless of where your script or an administrator initially puts your application, administrators are likely to move it elsewhere to suit their needs. Write code so that it will still run after being moved.*

Making the Distribution

For distributions on diskette, be sure to format each diskette with `fdformat` for high-density diskettes or `fdformat -L` for low-density diskettes before copying your files to them. The `fdformat`(8) `man` page contains more information.

The command syntax to place the `copyright`, `Installscript`, and your application files on high-density diskettes is:

```
bar cfb /dev/rfd0c 18 copyright Installscript
application_files
```

For low-density diskettes use:

```
bar cfb /dev/rfdl0c copyright Installscript
application_files
```

For tapes use:

```
bar cfb /dev/rst08 copyright Installscript
application_files
```

The `bar`(1) command is available for loading tapes and diskettes only on Sun386i systems. For tapes intended for other Sun systems, you must use `tar`(1) format:

```
tar cf /dev/rst08 copyright Installscript
application_files
```

How Users Will Load Your Software

Any user belonging to the `operator` group can use `snap` to load software that you package as described earlier in this section. (See *Sun386i SNAP Administration* for a description of groups.) Users will:

1. Insert the tape or diskette containing your software.
2. From the Desktop menu, pull right on Services and select the SNAP Administration option.
3. Place the mouse button on the cycle symbol and click right to display the Category menu.
4. Select the Software menu item.
5. Select the Unbundled Software item in the lower panel and click left on the Install button.
6. A pop-up window then appears, displaying your copyright and description file (which `snap` reads from the tape or diskette). At the end of this display, users are requested to select Confirm to load the software (the pop-up window also contains a Cancel selection).
7. The system starts a `shelltool`, in which it runs your installation script. Users cannot perform any other functions in `snap` or in the `shelltool` until the script completes. When it does, the system requests users to press any key to return to SNAP. The `shelltool` quits after a key is pressed.

10

Internationalizing Applications

10

Internationalizing Applications

An *internationalized product* is one built to allow easy and rapid modification to meet local country requirements. Such requirements involve the keyboard, code sets, date and time formats, currency symbols, collating sequences, message databases, and so on. A *localized product* is one that has been modified appropriately for a particular country. This chapter presents some guidelines for you to follow in your efforts to produce internationalized applications.

10.1. Internationalization Support

The Sun386i system provides full or partial support in the following internationalization areas:

- Clean handling of 8-bit characters in the kernel, Bourne shell, Text Editor, and DOS Windows
- Native-language messages in the SunOS kernel

The following sections discuss these topics in detail, emphasizing relevant features of the Sun386i system in particular.

8-Bit Characters

Most SunOS (3.x) software assumes 7-bit U.S. ASCII data. Many programs reject characters greater than 177 octal, or use the high-order bit for other purposes. Currently, the SunOS 4.0 kernel provides an 8-bit data path throughout. In addition, the Text Editor, DOS Windows, and Bourne shell utilities support 8-bit characters.

For applications, the Sun386i system provides a direct 8-bit data path from the keyboard. Applications can then use display fonts with 256 characters. Sun has provided one ISO font (`screen.iso.r.12`) and two international `dos` fonts (`pcfont.r.14` and `pcfont.b.14`, for regular and boldface, respectively). Both of these fonts are in the `/usr/lib/fonts/fixedwidthfonts` directory.

Note that at this time you cannot use 8-bit characters in file names or electronic mail messages. In addition, you cannot print these characters with the `lpr`(1) command.

Alternative Code Sets

The Sun386i system supports the three code sets described below.

- 7-bit U.S. ASCII — This is the same set currently used by Sun-3 products; it is the default code set.

- 8-bit International ASCII — This is the set specified by the ISO standard (8859/1); it's a superset of 7-bit U.S. ASCII.
- 8-bit IBM PC/DOS code set — This is the same set used on PCs, and is a superset of 7-bit U.S. ASCII.

Country distributors have access to code that allows the system to come up using 8-bit International ASCII instead of the default. It is up to these distributors to provide the appropriate keyboard mappings (see Keyboard Support below) as well as additional fonts. This is necessary because different countries have different keyboard layouts.

`dos` Issues

`dos`(1) uses the 8-bit MS-DOS code set. When a `dos` window is active, the system keyboard handler is superseded by a special keyboard handler that maps raw key codes into the MS-DOS code set. In this mode, Stop-a (abort sequence), for example, has no effect.

Because the MS-DOS character set is different from the ISO character set, you must convert ISO text files with the `unix2dos`(1) program to display them in DOS Windows. Similarly, to display text files containing MS-DOS international characters correctly in a SunOS Text Editor window, you must convert files with the `dos2unix`(1) utility. Copying and pasting between windows works correctly without any conversion. The options available with the `dos2unix`(1) and `unix2dos`(1) commands are described below.

`dos2unix -iso` and `unix2dos -iso` are required to ensure that MS-DOS international characters (such as umlauts and accents) are correctly converted to their ISO equivalents and vice versa. You should routinely include the `-iso` option with both commands. The MS-DOS characters with no direct counterparts under ISO are mapped to unused placeholders in the ISO character space. An ISO font will be unable to display these placeholders, but `unix2dos -iso` will correctly convert these characters back to the MS-DOS font without loss of information.

`dos2unix -7` is required only if you have MS-DOS graphics characters in a file and want to export that file to an application that doesn't support either 8-bit characters or one of the MS-DOS fonts. When `dos2unix -7` is used on a file, characters that have the eighth bit set are dropped.

For more information about these commands, refer to the `dos2unix`(1) and `unix2dos`(1) `man` pages. Translation tables in Appendix H show the character mapping that occurs when converting MS-DOS files to ISO format and vice versa.

Keyboard Support

A number of language-specific keyboards are available. This section discusses the Compose, Alt Graph, and floating accent keys, which enable display of additional international characters. The major differences between these keys from a distributor's perspective are:

- Country distributors must tailor their systems to provide the additional keys made possible via Alt Graph; Compose and floating accent key functionality is automatic.
- Compose key functionality is available with English keyboards, while Alt Graph and floating accent capability requires a country-specific keyboard.

Compose Key

You can press the Compose key, followed by two other keystrokes, to display various West European characters. Table 10-1 shows the keystrokes required to produce these characters. The leftmost character shows the end result — the character that you want to display. The two keys to the right are the ones to press after pressing the Compose key. For example, to display ä (a umlaut), press Compose-a-".

Table 10-1 *Compose Key Sequences*

NBSP	° ^*	À A‘	D D-	à a‘	đ d-
¡ !!	+ +-	Á A’	Ñ N~	á a’	ñ n~
¢ c/	2 ^2	Â A^	Ò O‘	â a^	ò o‘
£ L-	3 ^3	Ã A~	Ó O’	ã a~	ó o’
¤ ox	´ ’’	Ä A"	Ô O^	ä a"	ô o^
¥ Y-	µ /u	Å A*	Õ O~	å a*	õ o~
¦ ||	¶ P!	Æ AE	Ö O"	æ ae	ö o"
§ so	· ^.	Ç C,	x xx	ç c,	÷ –:
¨ ""	¸ ,,	È E‘	Ø O/	è e‘	ø o/
© co	1 ^1	É E’	Ù U‘	é e’	ù u‘
ª ^a	º ^o	Ê E^	Ú U’	ê e^	ú u’
« <<	» >>	Ë E"	Û U^	ë e"	û u^
¬ -|	1/4 14	Ì I‘	Ü U"	Ì i‘	ü u"
SHY --	1/2 12	Í I’	Ý Y’	Í i’	ý y’
® ro	3/4 34	Î I^	Þ P|	Î i^	þ p|
- ^-	¿ ??	Ï I"	ß ss	Ï i"	ÿ y"

The only change that country distributors need make to ensure that the Compose key works properly is to use the `defaults` utility to reset the default font to `/usr/lib/fonts/fixedwidthfonts/screen.iso.r.12` or to any valid ISO font available.

Note that NBSP (no-break space) and SHY (soft hyphen) in Table 10-1 are nonprinting characters for controlling line and word breaks in applications.

Alt Graph Key

The Compose key enables display of many but not all international characters. You can display the third character that appears in the lower right corner on some international keycaps by using the Alt Graph key. Alt Graph works similarly to the Shift key — simultaneously press Alt Graph and the key containing the character to display that character.

Country distributors can enable Alt Graph capability by performing the following steps:

1. Remount `/usr` as writable by becoming root and entering the command:

   ```
   mount -o remount,rw /usr
   ```

2. Create the file `/usr/lib/keymaps/`*country_name*`.keys`, specifying:

   - How this key must be used in the first column — specify `BASE` for keys pressed alone, `SHIFT` for keys pressed while the Shift key is pressed, `CAPS` for keys pressed after the Caps key is pressed, and `ALTG` for keys pressed while the Alt Graph key is pressed
   - The character's decimal keystation location (shown in Figures 10-1 and 10-2 on the next page) in the second column
   - The hexadecimal ISO code corresponding to that character in the third column

   Use # to indicate comments, and separate columns with a space.

2. Use the `setkeys`(1) command with the format:

   `setkeys -f /usr/lib/keymaps/`*country_name*`.keys`.

3. Redefine the default font to `/usr/lib/fonts/fixedwidthfonts/screen.iso.r.12` or to any valid ISO font available.

4. Remount `/usr` as read only by becoming root and entering the command:

   ```
   mount -o remount,r /usr
   ```

Figure 10-1 shows the U.S. keystation map and Figure 10-2 shows the international keystation map (both are on the next page). Table 10-2 on page 154 contains the ISO codes needed. The file `/usr/lib/keymaps/canada.keys` shows a sample file for the Canadian keyboard.

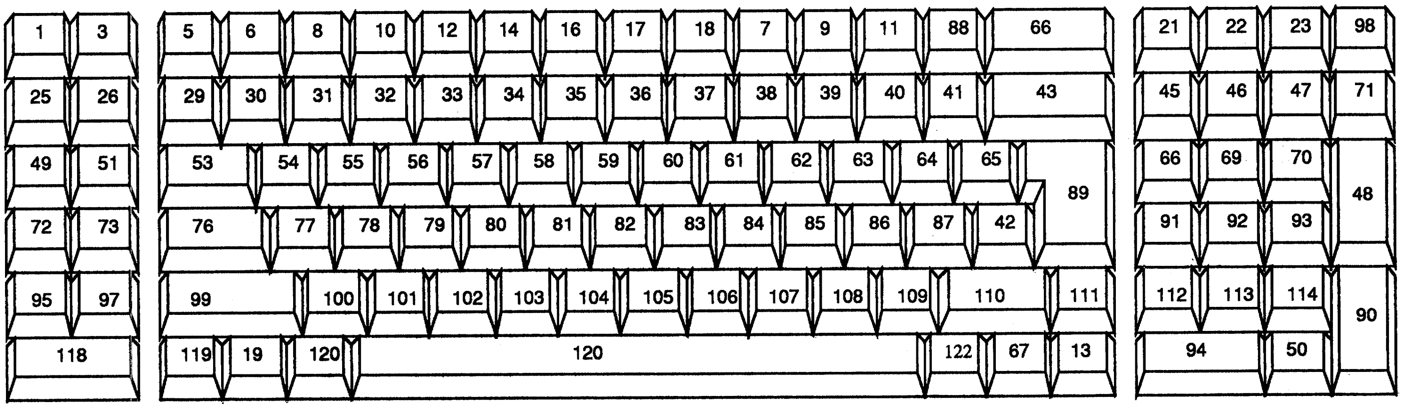


Figure 10-1 *U.S. Keystation Map*

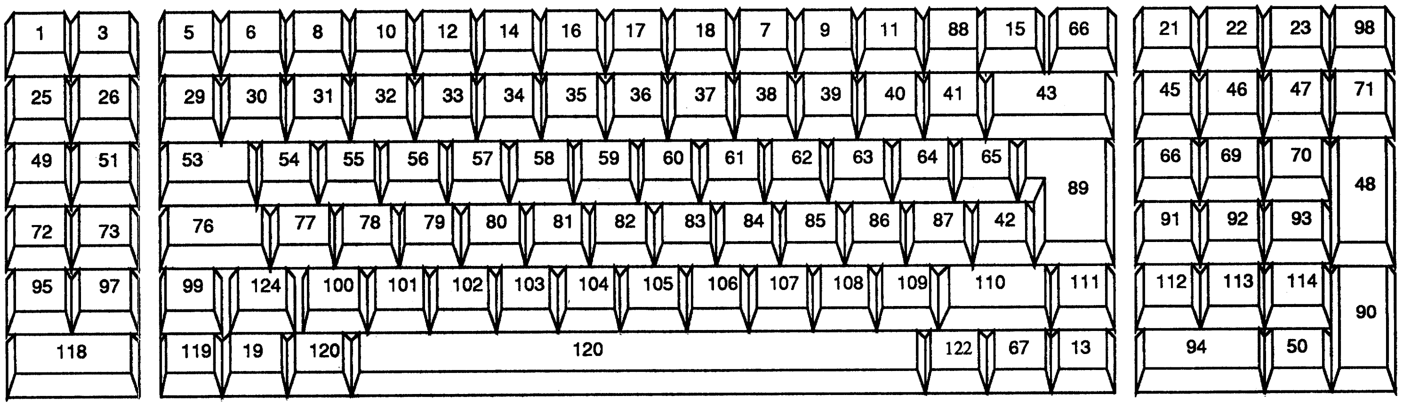


Figure 10-2 *International Keystation Map*

Table 10-2 *ISO Character Set*

	Upper Nibble															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Lower Nibble																
00			SP	0	@	P	‘	p			NBSP	°	À	Đ	à	đ
01			!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
02			"	2	B	R	b	r			¢	2	Â	Ò	â	ò
03			#	3	C	S	c	s			£	3	Ã	Ó	ã	ó
04			$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
05			%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
06			&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
07			’	7	G	W	g	w			§	·	Ç	x	ç	÷
08			(	8	H	X	h	x			¨	¸	È	Ø	è	ø
09			)	9	I	Y	i	y			©	1	É	Ù	é	ù
0A			*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
0B			+	;	K	[	k	{			«	»	Ë	Û	ë	û
0C			,	<	L	\	l	|			¬	1/4	Ì	Ü	ì	ü
0D			-	=	M	]	m	}			SHY	1/2	Í	Ý	í	ý
0E			.	>	N	^	n	~			®	3/4	Î	Þ	î	þ
0F			/	?	O	_	o				-	¿	Ï	ß	ï	ÿ

Floating Accent Key

You can create accent characters with the floating accent key available on international keyboards. The floating accent key works similarly to the Compose key, in that you press the floating accent key and then the character that you want to accent. You can accent both lower- and uppercase characters; to accent the latter, press the floating accent key, the Shift key, and then the character that you are accenting.

Native-Language Messages

To aid in the translation process, the Sun386i system manages external message libraries and routines for selecting and accessing system messages and on-screen help text, as described below.

System Messages

System messages are those originating in the SunOS kernel and system daemons. The `syslogd`(8) daemon intercepts error messages and uses them as keys into a message database. The replacement messages, not the original ones, are displayed on the consoles. Country distributors can provide translations by:

- Translating the file `/etc/Out` to the desired language
- Using the `kill -1` command on the `syslogd` process to activate the changes made

To translate additional messages that Sun has not reworded requires the ability to determine the original message text, usually by looking through source code. If you have a license permitting this, you might want to translate additional messages as described in the Kernel Error Messages section (starting on page 73).

On-Screen Help Text

On-screen help files, both Spot Help and Help Viewer handbooks, are also arranged for easy translation. Table 6-1 on page 88 provides a list of the help files that come with the Sun386i system and their locations.

10.2. Application Guidelines

This section contains guidelines to help you internationalize your applications and increase the potential for international sales. Note that Sun software does not yet conform to some of these guidelines.

8-Bit Characters

To enable use of ISO fonts, don't use the eighth bit for a flag.

Don't use nonASCII text characters unless you want them to be displayed.

Avoid input syntax using special characters such as `[ ] { } # @ ~` and ^.

Date and Time Formats

Be aware that European countries have a different format for representing dates. For example, July 24, 1987 would be represented in the U.S. as `7/24/87`, but in Europe as `24/7/87` (comma, hyphen, period, or space separators are also used) or `19870724` (metric format).

Allow times to be specified in 24-hour format. For example, `4:00 PM` is `16:00` or `1600`.

Numeric Formats

Allow input and display of positive and negative numbers with the + and – signs after the quantity, for example `10-`.

Allow input and display using an apostrophe, period, or space as the thousands' separator in addition to the comma conventionally used in the U.S. Also, allow the use of a comma as the separator between the integer and decimal amounts. For example, `2,311.50` would typically be represented in Europe as `2.311,50`.

Currency Symbols

Allow for multicharacter currency symbols and their placement either before or after the amount. For example, twenty Danish kroner are represented as `20 DKr`.

Text Messages

Use a separate text file for messages if possible, or put all `printf`(3S) statements in a separate module.

Include prompts and responses in a text file/module. Prompts must be readily modifiable to work with the target language.

Document where the message text is located and how to access it.

Allow multiple message catalogs to coexist, to let individual users choose their preferred language.

Structure your message files/modules and screen message presentations so that they can be easily expanded (by up to 50%) to accommodate the greater space requirements that foreign languages typically have.

A

Sun386i System Description

A

Sun386i System Description

This appendix provides a complete description of Sun386i system features, at both the technical and user interface levels, as well as a summary of available hardware and software.

A.1. Product Features

The basic system consists of a system unit holding the power supply and internal electronics, one full-height 91 Mbyte hard disk drive, and one half-height diskette drive. This unit can operate as a diskless node without mass storage devices and as a server.

You can add an expansion unit to the system that can house a power supply, one full-height hard (327 Mbyte) disk drive, and one streaming tape drive. Optionally, the expansion unit can include a SCSI tape/disk controller card.

System Unit

The system unit is made up of the following submodules:

- Power supply
- CPU board/backplane
- 3 1/2-inch diskette drive
- 5 1/4-inch hard disk drive (full or half-height)

The system unit can lie flat in desktop locations or stand vertically (on a separate base) in deskside locations.

The CPU board has eight slots:

1. System bus slot — 4, 8, or 16 Mbyte memory board
2. System bus slot — 4, 8, or 16 Mbyte memory board
3. System bus slot — 4, 8, or 16 Mbyte memory board
4. System bus slot — monochrome or color frame buffer, or 4, 8, or 16 Mbyte memory board
5. AT slot — unallocated, user-configurable (16-bit device)
6. AT slot — unallocated, user-configurable (16-bit device)
7. AT slot — unallocated, user-configurable (16-bit device)
8. XT slot — unallocated, user-configurable (8-bit device)

The System bus is a proprietary, high-speed, 32-bit bus. Maximum system memory is 16 Mbytes.

The following input/output ports are available at the rear of the system unit:

- Ethernet — 15-pin Dsub (female)
- PC-compatible parallel output port — 25-pin Dsub (female)
- PC-compatible RS-423 serial I/O port — 25-pin Dsub (male)
- External SCSI port — 50-pin connector (female)

Cooling is by forced convection in the front-to-rear direction. The fans cool the PC boards and power supply as well as the optional diskette and hard disk drives.

The power supply is a semi-custom design and plugs directly into the CPU board. The power supply itself is a fully enclosed submodule that contains the power switch, AC inlet, courtesy outlet, and voltage select switch. The power supply includes power for the 15-inch monochrome monitor only; other monitors must be plugged directly into the facility's power source. The courtesy outlet on the main power supply is only for use with the expansion unit.

Expansion Unit

The optional expansion unit attaches to the system unit, thus giving one unit assembly. When fastened together, they form a deskside configuration (with a floor stand). The signal cable running between the expansion and system units is external.

The expansion unit includes the following submodules:

- Power supply submodule
- Two full-height 5 1/4-inch peripherals
- Optional SCSI controller

The power supply submodule contains a 120 W power supply. Like the CPU supply, it is a self-contained, fully encased unit. The rear of the supply has an AC inlet plug, an AC power switch, and a voltage select switch. This submodule also contains a fan. The combined acoustic limit for both the system unit and the expansion unit is 50 dB(A), to conform to German standards.

The peripheral locations have the ability to mount either full- or half-height devices. Only one full-height or two half-height devices can have removable media accessible through the front of the enclosure.

Cooling is by forced convection in the front-to-rear direction.

AT Bus

The AT bus interface couples the 32-bit Sun386i system to an AT bus structure. The AT bus, along with the MS-DOS operating system, enables the Sun386i system to run PC applications. The major components of the interface are:

- Data buffer and alignment logic — 32-bit to 8- or 16-bit data conversion
- Address buffers — passes least significant 18 address lines; generates `A0`, `A1`, and `BHE`. Master mode will 3-state these lines.

- DMA and interrupt logic — connects the 7 AT bus DMA channels to 7 system DMA channels; connects the 11 AT bus interrupts to 11 system interrupt channels
- Signal generation and control logic — selectable timings based on a harmonic of the system clock

Chapter 8 provides additional information about the AT bus.

Monitors

The system is compatible with existing monochrome 19-inch and color 19-inch Sun monitors. In addition, 15-inch monochrome and 16-inch color monitors are available.

Keyboard and Mouse

The low-profile, microprocessor-controlled keyboard accepts an optical mouse. The keyboard connects to the CPU via a coiled cord. This is a portion of the monitor-keyboard interface cable.

For each key on the Sun-3 keyboard there is a corresponding key on the new keyboard, though not necessarily in the same position. The corresponding key is labeled similarly and causes generation of the same code.

The keyboard is a superset of the AT-style (84-key) and the Sun-3 keyboards. In addition, the keyboard contains Help, Alt Graph, and Compose keys. The Help key is used for displaying cursor-sensitive help messages provided with the SunView window facilities. The Compose key allows composition and use of a series of West European characters which would not otherwise be available.

The Compose key enables display of many but not all additional international characters. Users can display the third character that appears on some international keycaps by using the Alt Graph key. Additionally, users can display accented characters by using the floating accent key, available on international keyboards.

The keys are layed out in three keypad areas:

- 10 Sun "L" keys on the left
- The main keyboard area in the center
- A keypad area functioning as the 15 Sun "R" keys, plus the keys normally found on the right block of an AT keyboard, on the right

In addition, 15 "F" keys are located across the top of the center keypad. These keys represent the 9 Sun function keys in Sun mode or 12 function keys in AT mode.

The keyboard includes lights to indicate shift, scroll lock, numeric lock, and compose, and an internal speaker to generate key clicks each time a key is pressed. You can enable or disable this click from either a unique sequence of keystrokes or a command sent to the keyboard from the host CPU.

The keyboard also can generate a beep on command from the host CPU. The volume, pitch, and duration of this beep is adjustable by using either a unique sequence of keystrokes on the keyboard or a command generated by the host CPU.

Mass Storage Devices

The main enclosure can be configured with one half-height 3 1/2-inch 1.44 Mbyte diskette drive plus one full-height 5 1/4-inch 91 or 327 Mbyte formatted hard disk drive. The following mass storage devices are available for installation in the main housing:

- 91 Mbyte formatted SCSI Winchester
- 327 Mbyte formatted SCSI Winchester
- One half-height, AT-compatible 1.44 Mbyte 3.5-inch diskette

The following devices are supported in the expansion unit:

- 327 Mbyte formatted SCSI Winchester
- SCSI compatible/Sun-compatible streaming tape

Dimensions and Weights

Dimensions and weights are provided in the following table.

Table A-1 *Dimensions and Weights*

Component	***Height***	***Width***	***Depth***	***Weight***
System Unit	7" (17.8 cm)	15" (38.1 cm)	20" (50.8 cm)	45 lbs (20.4 kg)
Expansion Unit	7" (17.8 cm)	8" (20.3 cm)	20" (50.8 cm)	25 lbs (11.4 kg)
Keyboard	2" (5.1 cm)	19" (48.3 cm)	8" (20.3 cm)	2 lbs (0.9 kg)
15-inch Monitor	13" (33.0 cm)	15" (38.1 cm)	13" (33.0 cm)	25 lbs (11.4 kg)
Mouse	2" (5.1 cm)	4" (10.2 cm)	3" (7.6 cm)	0.3 lbs (0.14 kg)

Electrical Power Requirements

Each system requires one AC receptacle for the system unit. When the expansion unit is attached, it plugs into the system unit's courtesy outlet. Monitors other than 15-inch monochrome plug into the facility's power source.

Table A-2 *Electrical Power Requirements*

	Voltage	***Input Power***	***Frequency***
System Unit	90 – 132V ac 180 – 264V ac	400 W	47 – 63 Hz
Expansion Unit	90 – 132V ac 180 – 264V ac	200 W	47 – 63 Hz
15-inch Monitor	from system unit	50 W	

Input voltage is switch-selectable at the rear of the modules.

Available power supply outputs are as follows:

- System Unit
  - +5V dc ok signal
  - +5V dc @ 31/38 amps
  - +12V dc @ 4.3 amps / 5.6 amps peak
  - –12V dc @ 0.5 amps
  - –5.2V dc @ 1.9 amps
  - +80V dc @ .625 amp if +5vdc load < 31 amps

- Expansion Unit
  - +5V dc @ 6 amps
  - +12V dc @ 6.5/12 amps pk

Environmental Requirements

Environmental requirements are listed in the following tables.

Table A-3 *Operating Environment Requirements*

Temperature	0°C to 40°C (32°F to 104°F)
Humidity	5 to 80% relative noncondensing (@ 40°C)
Wet Bulb	25°C (77°F) maximum
Altitude	0 m to 2134 m (0 to 7000 ft)
Vibration	5 – 22 Hz, 0.01 inches p-p
	22 – 500 Hz, 0.25 g pk
Shock	5 g peak, 10 msec 1/2 sinewave

Table A-4 *Nonoperating Environment Requirements*

Temperature	–20°C to 60°C (–4°F to 140°F)
Humidity	5 to 90% relative noncondensing (@ 40°C)
Wet Bulb	46°C (115°F) maximum
Altitude	0 m to 12,192 m (0 to 40,000 ft)
Vibration	5 – 22 Hz, 0.02 inches p-p
	22 – 500 Hz, 0.5 g peak
Shock	20 g pk, 30 msec

A.2. Hardware

Primary hardware components include a CPU board, a monochrome or color frame buffer, and up to four memory boards (systems with frame buffers can have a maximum of three memory boards).

CPU Board

The CPU board has the following features:

- Intel 80386 CPU, 20 to 25 MHz
- Intel 80387 numeric coprocessor
- Intel 82380 32-bit integrated DMA, interrupt, and timer controller
- Intel 82586 Ethernet controller
- Western Digital WD33C93-based SCSI host adapter
- One RS-423 serial I/O port, PC-compatible
- One PC-compatible parallel output (printer) port
- 128 Kbyte PROM
- 2 Kbyte "Zero Power" RAM/TOD chip
- ID PROM

- Speaker output (signal generated by 82380)
- One XT slot, three AT/XT slots, and four System bus (proprietary) slots

Frame Buffers

Both monochrome and color frame buffers are available, as described in this section. Chapter 3 provides additional frame buffer information.

Monochrome Frame Buffer

The monochrome frame buffer has a resolution of 1152x900x1 and supports existing Sun monitors. It includes the interface for the keyboard and mouse.

Color Frame Buffers

Two color frame buffers are available. The first is a 1024x768x8 frame buffer compatible with the 1024x768 color monitor used with this system. The second is a 1152x900x8 frame buffer compatible with the existing Sun 19-inch color and 19-inch grayscale monitors. Both frame buffers include connections for the keyboard and mouse.

The color frame buffers interface to the CPU board via the 32-bit System bus. It is based on the Brooktree DAC and provides 1 Mbyte of 250 ns color/grayscale video memory. Any 256 of a total of 16 million possible colors are available at one time. These frame buffers do not provide a monochrome or enable plane.

SIMM Memory Board

The Sun386i system uses Single In-line Memory Module (SIMM) boards, which use the Intel 82385 cache controller chip. SIMM boards contain 16 slots, each of which can hold a 1 Mbyte SIMM module. The SIMM board comes equipped with 8 Mbytes of memory but you can expand it to 16 Mbytes by adding additional SIMM modules. Each system can have only one SIMM board.

A.3. Diagnostics

Three classes of diagnostics are available, as described in the following sections.

Power-Up Diagnostics

The first class is the power-up diagnostics. These consist of two components: one component is run out of the boot ROM and the other is loaded into the system after the boot ROM diagnostics have verified the boot path. Both components are run whenever power is applied to the system. They run in less than 30 seconds and verify system operation to a confidence level of 80%.

Hardware Diagnostics

The second class of diagnostics is the standalone hardware diagnostics. This class consists of Hardware Diagnostics, user-oriented diagnostics that are part of Application SunOS (described on the next page) and the Diagnostic Executive™. The operator can run Hardware Diagnostics any time after completion of power-up diagnostics. (However, the operator must first shut down the system before running the diagnostics if the operating system is already running.) A single pass through Hardware Diagnostics takes less than 10 minutes and verifies system operation to a confidence level of 95%. The Diagnostic Executive is for more experienced users, and is an unbundled product.

System Exerciser

The third class of diagnostics is the System Exerciser. (These are also part of Application SunOS, described on the next page.) The System Exerciser runs under the SunOS system and verifies operation of the total system, including operating system software. It includes tests of physical memory, virtual memory, mass storage devices, frame buffer, floating point support, and Ethernet support.

A.4. Operating System

The Sun386i system uses version 4.0 of the Sun Operating System (SunOS). This operating system comprises Berkeley Release 4.2 with some 4.3 features plus enhancements to provide support of the System V.3 Interface Definition (SVID). System software is divided into two unbundled subsets—Application SunOS and SunOS Developer's Toolkit. Chapter 9 describes these divisions in greater detail.

Application SunOS

Application SunOS is itself divided into four subsets:

- Hardware Diagnostics — standalone hardware diagnostics, described in the preceding section
- Core system — the base system, providing the ability to run most commercially available Sun and third-party applications; the core system is shipped on the Sun386i disk
- Optional clusters — additional software for capabilities such as extended mail and extended networking; available on diskettes or tape
- Recovery software — a backup version of the core system, available on diskettes or tape

Buying Application SunOS is not mandatory, but users are strongly urged to do so. By purchasing the package, they receive all of the software shown above. When all optional clusters are loaded, Application SunOS takes up about 14 Mbytes. Application SunOS is available on 3 1/2-inch diskettes and 1/4-inch tapes.

SunOS Developer's Toolkit

SunOS Developer's Toolkit is a complement to Application SunOS. It is intended for developers and others who require the full capabilities of the Sun operating system. It contains everything in the SunOS 4.0 system that is not included in Application SunOS.

The Developer's Toolkit occupies about 18 Mbytes of disk space and is available on diskettes or 1/4-inch cartridge tapes. Chapter 2 describes installation of the Developer's Toolkit.

A.5. Languages

The assembler, C compiler, shared libraries, other language tools (such as the link editor, profilers, and so on), and debuggers (`adb`, `dbx`, `kadb`) are part of the Developer's Toolkit. FORTRAN and Pascal are packaged and sold separately, and require the presence of Developer's Toolkit on the system. All languages are available on 1/4-inch streaming tape and support the 80387 numeric coprocessor.

A.6. Windows and Graphics

The Sun386i system includes software to support windows and graphics as described below.

Pixrects

Programmers can create graphic images on the frame buffers through a Pixrects interface which is identical (at the level of the calling functions) to that provided for Sun-3 products.

SunView

The Sun386i system includes the SunView windowing system. From the perspective of the calling routines and the system user, SunView on the Sun386i system is similar to SunView provided for the Sun-3 product line.

SunView Applications

The Sun386i system also includes the set of desktop management tools known as SunView applications. These are identical to those offered on Sun-3 products, with the exception of some additions (`dos(1)`—part of the MS-DOS facility; `snap(1)`—part of the new system administration tools; `organizer(1)`—a graphical file system window interface; `help_viewer(1)`—a program providing on-screen information about the system). Chapter 6 describes `snap`, `organizer`, and `help_viewer` in more detail. Chapter 7 describes `dos`.

Users with color systems can use the `coloredit(1)` facility to add or change the foreground and background colors of SunView applications. Programmers can add color to panels and panel items within their applications. Chapter 6 describes both of these color capabilities in more detail.

A.7. Unbundled Software

In addition to FORTRAN and Pascal, both mentioned in Section A.5 on the previous page, the following unbundled software is available:

- SunCGI
- SunGKS
- SunUNIFY
- SunSimplify

A.8. MS-DOS Compatibility

The system provides the capability to run any program that runs on an IBM AT under the MS-DOS 3.3 operating system. This capability is based on PC emulation and the inclusion of MS-DOS on the system. MS-DOS is available in special MS-DOS windows (DOS Windows; the program name is `dos`), and MS-DOS commands can run in SunOS windows.

The MS-DOS PC emulation capability includes the following features:

- Virtual hard disk emulation using either the hard disk on the host machine or one available through the network
- The ability to read and write to the diskette drive on the host system
- Special MS-DOS windows (`dos`) for running of MS-DOS programs
- The ability to run multiple MS-DOS programs by running each in a separate window. However, only one of the windows will allow a program access to any individual device on the AT bus; the other MS-DOS processes are prevented from accessing these devices.
- MS-DOS program access to files created by SunOS programs and vice versa
- The ability to cut and paste text between MS-DOS and other SunView windows
- MS-DOS program access to an 8087 numeric coprocessor, emulated via the 80387 numeric coprocessor
- Microsoft Mouse emulation (with the standard system mouse)
- PC-compatible serial and parallel port emulation
- Emulation of the Monochrome Display Adapter (MDA), Color Graphics Adapter (CGA), and Hercules graphics card

Chapter 7 in this manual contains additional information about MS-DOS on the Sun386i system.

A.9. Administration Tools

The Sun386i system incorporates new system administration tools, including a window-based application for bitmapped systems called `snap`(1). These tools provide the following capabilities:

Automatic System Installation
This feature enables a user to have the system up and running on an existing Sun386i system network within 30 minutes after unpacking.

Support for Creating New User Accounts
This capability involves two areas:

- New user access, which allows a user without a log-in name on a system to follow displayed directions to create an account
- Full screen login, whereby users entering their user names and passwords can view help screens by pressing the [Help] key.

Backup Facilities
This capability provides backup support that provides system-wide and personal backup functions.

Chapter 6 provides additional information about the administration facilities just described.

A.10. User Interface

Some of the new user interface features on the Sun386i system are:

- An on-screen help facility (described in the next section)
- The `coloredit`(1) color editing facility (described on the previous page)
- Ease-of-use administration features, including the `snap`(1) utility (described in the preceding section)
- The `organizer`(1) file system utility (described on the preceding page)

Chapter 6 contains additional information about all of these features.

A.11. Documentation

Both on-line and hardcopy information is available for the Sun386i system.

On-Line Documentation

The Sun386i system offers on-line documentation in four areas:

- Log-in screens, to facilitate the novice user's entry into the system
- Help windows, popped up by pressing the [Help] key, to provide cursor-sensitive help in an alert box and access to Help Viewer, a hypertext-based on-screen documentation system
- System (kernel) messages in terms users can understand
- Traditional SunOS `man` pages (also available in hardcopy), changed to reflect features specific to the Sun386i system; `man` pages are part of the optional `man_pages` cluster of Application SunOS. (Chapter 9 describes the division of system software and clusters.) Appendix G lists the `man` pages that are new or changed for the Sun386i system, as well as the SunOS 4.0 pages that do not pertain to the Sun386i system.

Printed Documentation

Printed documentation for the Sun386i system includes four sets of manuals.

The *Sun386i Owner's Set* includes four manuals and one pamphlet specific to the Sun386i system and directed to an end-user audience. They are intended for users whose primary interest is in running applications. One set is shipped with every system.

The *Sun386i Owner's Supplement Documentation Set* contains 10 titles, including `man` pages, that collectively form the end-user documentation for all Sun systems. Users receive this set if they purchase Application SunOS.

The *Sun386i Developer's Toolkit Documentation Set* includes 12 titles, all of which are programming guides directed at application developers and system programmers on all Sun systems. This set includes this manual, and is shipped to every customer who purchases the Developer's Toolkit optional software.

The *Sun386i Upgrade Documentation Set* includes three technical manuals and two sets of release notes specific to the Sun386i system. The upgrade set is a complement to the documentation set for SunOS 4.0.

Additional documentation includes the *Sun386i Technical Overview* and the *Sun386i Field Service Manual.* Both must be ordered separately. Documentation for FORTRAN and other unbundled software is shipped with those products.

A.12. Internationalization

An internationalized product is one built to allow easy and rapid modification to meet local country requirements. Such requirements involve the keyboard, character sets, time/date formats, currency symbols, collating sequences, and so on. This is different from a localized product. A localized product is an internationalized product that has been appropriately modified for a particular country.

The specific internationalization features of the Sun386i system are stated below. Except for the provision of local language keyboards, the Sun386i system is not a localized product.

- Local language keyboards are available for some countries outside of the United States.
- Software support is provided for new keyboards to ensure that pressing local language keys does not have an unexpected affect. Specifically, most SunView windows ignore 8-bit codes; the exceptions are Text Editor, DOS Windows, and the Bourne shell, which do accept 8-bit codes.
- Applications can receive 8-bit codes from the keyboard; subsequent processing, display, and printing are the responsibility of the application.
- 8-bit data will not be corrupted by the SunOS kernel, window system, or shell. Note that this requirement applies to the data only; no provision exists to support file names or program names using 8-bit characters.

The above is **not** a fully internationalized product; rather, it reflects an effort to meet the minimum needs of application vendors in the international market.

B

80386 Assembly Language Definition

B

80386 Assembly Language Definition

This appendix is only slightly modified from a draft provided by AT&T. It is their third draft of the assembler language definition for the 5.3/386 CCS. Being preliminary in nature, it is subject to change. If you are doing assembly language programming, you should also obtain a copy of Intel's *80386 Programmer's Reference Manual* (Intel Corporation, Santa Clara, CA).

The Sun386i C compiler only uses a limited number of the symbols, expressions, pseudo operations, and machine instructions described in this appendix. However, the entire set is included to give a complete description of the as(1) assembler, which accepts all of them.

B.1. Invoking the Assembler

For information about invoking the Sun386i assembler and the available options, refer to the as(1) man page in the *SunOS Reference Manual.*

Input Format

The input to the assembler is a text file. This file must consist of a sequence of lines ending with a newline character (ASCII LF). Each line can contain one or more statements. If several statements appear on a line, they must be separated by semicolons (;). Each statement must be one of the following:

- An empty statement is one that contains nothing other than spaces, tabs, and form-feed characters. Empty statements have no meaning to the assembler. They can be inserted freely to improve the appearance of a listing.
- An assignment statement is one that gives a value to a symbol. It consists of a symbol, followed by an equal sign (=), followed by an expression. The expression is evaluated and the result is assigned to the symbol. Assignment statements do not generate any code. They are used only to assign assembly time values to symbols.
- A pseudo operation statement is a directive to the assembler that does not necessarily generate any code. It consists of a pseudo operation code, optionally followed by operands. Every pseudo operation code begins with a period (.).
- A machine operation statement is a mnemonic representation of an executable machine instruction that is translated by the assembler. It consists of an operation code, optionally followed by operands.

In addition, you can modify each statement by doing one or both of the following:

- Place a label at the begining of any statement. This consists of a symbol followed by a colon (:). When the assembler encounters a label, it assigns the value of the location counter to the label.
- Insert a comment at the end of any statement by preceding the comment with a slash (/). The assembler ignores characters following a slash on a line. This facility is provided to allow insertion of internal program documentation into the source file for a program.

Output Format

The output of the assembler is an object file. The object file produced by the assembler contains at least the following three sections:

`.text` This is an initialized section; normally it is read only and contains the code from a program. It may also contain read-only tables.

`.data` This is an initialized section; normally it is readable and writable. It contains initialized data. These can be scalars or tables.

`.bss` This is an uninitialized section. Space is not allocated for this segment in the object file.

An optional section, `.comment`, may also be produced (see Pseudo Operations on page 175).

Every statement in the input assembly language program that generates code or data places it into one of these three sections. The section into which the generated bytes are written starts out as `.text`, but you can change this by using section control pseudo operations.

B.2. Symbols and Expressions

This section describes the symbols and expressions that the assembler uses.

Values

Values are represented in the assembler by 32-bit, two's complement values. All arithmetic is performed using 32 bits of precision. Note that the values used in an 80386 instruction may use 8, 16, or 32 bits.

Types

Every value is an instance of one of the following symbol types:

undefined
An undefined symbol type is one whose value has not yet been defined. Examples of undefined symbol types are forward references and externals.

absolute
An absolute symbol type is one whose value does not change with relocation. Examples of absolute symbol types are numeric constants and expressions whose operands are only numeric constants.

text A text symbol type is one whose value is relative to the `.text` segment.

data A data symbol type is one whose value is relative to the `.data` segment.

bss A bss symbol type is one whose value is relative to the `.bss` segment.

You can give any of these symbol types the attribute EXTERNAL.

Symbols

A symbol has a value and a type, each of which is either specified explicitly by an assignment statement or implicitly from context. Refer to the Expressions section, which follows, for the regular expression definition of a symbol.

The following symbols are reserved by the assembler:

. Commonly refered to as dot. This is the location counter while assembling a program. It takes on the current location in the text, data, or bss section.

.text This symbol is of type text. It is used to label the beginning of a .text section in the program being assembled.

.data This symbol is of type data. It is used to label the beginning of a .data section in the program being assembled.

.bss This symbol is of type bss. It is used to label the beginning of`a .bss section in the program being assembled.

Expressions

The expressions accepted by the SunOS assembler can be described by their semantic and syntactic rules.

The following are the operators supported by the assembler:

Operator	***Action***
+	Addition
–	Subtraction
*	Multiplication
\/	Division
&	Bit-wise logical and
|	Bit-wise logical or
>	Right shift
<	Left shift
\%	Remainder operator
!	Bit-wise logical and not

Syntactic Rules for the Assembler

In the following syntactic rules, the nonterminals are represented by lowercase letters, the terminal symbols are represented by uppercase letters, and the symbols enclosed in double quotes are terminal symbols.

There is no precedence to the operators. You must use square brackets to establish precedence.

```
expr    : term
        | expr "+" term
        | expr " term
        | expr "/" term
```

```
                | expr "&" term
                | expr "|" term
                | expr ">" term
                | expr "<" term
                | expr "" term
                | expr "!" term
                | expr "-" term
                ;

term            : id
                | number
                | "-" term
                | "[" expr "]"
                | "<o>" term
                | "<s>" term
                ;

id              : LABEL
                ;

number          : DEC_VAL
                | HEX_VAL
                | OCT_VAL
                | BIN_VAL
                ;
```

You can describe the terminal nodes by using the following regular expressions:

```
LABEL   = [a-zA-Z_][a-zA-Z0-9_]*:
DEC_VAL = [1-9][0-9]*
HEX_VAL = 0[Xx][0-9a-fA-F][0-9a-fA-F]*
OCT_VAL = 0[0-7]*
BIN_VAL = 0[Bb][0-1][0-1]*
```

In the above regular expressions, choices are enclosed in square brackets; a range of choices is indicated by letters or numbers separated by a dash (–); and the asterisk (*) indicates zero or more instances of the previous character.

Semantically, the expressions fall into two groups, absolute and relocatable. The equations later in this section show the legal combinations of absolute and relocatable operands for the addition and subtraction operators. All other operations are only legal on absolute-valued expressions.

All numbers have the absolute attribute. Symbols used to reference storage, text, or data are relocatable. In an assignment statement, symbols on the left side inherit their relocation attributes from the right side.

In the equations below, `a` is an absolute-valued expression and `r` is a relocatable-valued expression. The resulting type of the operation is shown to the right of the equal sign.

```
a + a = a
r + a = r
a - a = a
r - a = r
r - r = a
```

In the last example, you must declare the relocatable expressions before taking their difference.

Following are some examples of valid expressions:

```
label
$label
[label + 0x100]
[label1 - label2]
$[label1 - label2]
```

Following are some examples of invalid expressions:

```
[$label - $label]
[label1 * 5]
(label + 0x20)
```

B.3. Pseudo Operations

Below is a list of the pseudo operations supported by the assembler. This is followed by a separate listing of pseudo operations included for the benefit of the debuggers `sdb` (not supported on the Sun386i system) and `dbx`(1).

General Pseudo Operations

`.align val`

The `.align` pseudo op causes the next data generated to be aligned modulo `val`. `val` must be a positive integer value.

`.bcd val`

The `.bcd` pseudo op generates a packed decimal (80-bit) value into the current section. This is not valid for the `.bss` section. `val` is a nonfloating-point constant.

`.bss`

The `.bss` pseudo op changes the current section to `.bss`.

`.bss tag, bytes`

Define symbol `tag` in the `.bss` section and add `bytes` to the value of dot for `.bss`. This does not change the current section to `.bss`. `bytes` must be a positive integer value.

`.byte val [,val]`

The `.byte` pseudo op generates initialized bytes into the current section. This is not valid for `.bss`. Each `val` must be an 8-bit value.

`.comm name, expr`
: The `.comm` pseudo op allocates storage in the `.data` section. The storage is referenced by the symbol `name`, and has a size in bytes of `expr`. `expr` must be a positive integer. `name` cannot be predefined.

`.data`
: The `data` pseudo op changes the current section to `.data`.

`.double val`
: The `.double` pseudo op generates an 80287/387 long real (64 bits) into the current section. Not valid in the `.bss` section. `val` is a floating point constant.

`.even`
: The `.even` pseudo op aligns the current program counter (`.`) to an even boundary.

`.float val`
: The `.float` pseudo op generates an 80287/387 short real (32 bits) into the current section. This is not valid in the `.bss` section. `val` is a floating-point constant.

`.globl name`
: This pseudo op makes the variable `name` accessible to other programs.

`.ident string`
: The `.ident` pseudo op creates an entry in the comment section containing `string`. `string` is any sequence of characters, not including the double quote (").

`.lcomm name, expr`
: The `.lcomm` pseudo op allocates storage in the `.bss` section. The storage is referenced by the symbol `name`, and has a size of `expr`. `name` cannot be predefined, and `expr` must be a positive integer type.

`.long val`
: The `.long` pseudo op generates a long integer (32-bit, two's complement value) into the current section. This pseudo op is not valid for the `.bss` section. `val` is a nonfloating-point constant.

`.noopt`
: The `.noopt` pseudo op.

`.optim`
: The `.optim` pseudo op.

`.set name, expr`
: The `.set` pseudo op sets the value of symbol `name` to `expr`. This is equivalent to an assignment.

`.string str`
: This pseudo op places the characters in `str` into the object module at the current location and terminates the string with a null. The string must be enclosed in double quotes (""). This pseudo op is not valid for the `.bss` section.

`.text`
: The `.text` pseudo op defines the current section as `.text`.

`.value expr [,expr]`

The `.value` pseudo op is used to generate an initialized word (16-bit, two's complement value) into the current section. This pseudo op is not valid in the `.bss` section. Each `expr` must be a 16-bit value.

`.version string`

The `.version` pseudo op puts the C compiler version number into the `.comment` section.

sdb Pseudo Operations

The Sun386i system does not support the `sdb` debugger; however, the assembler does recognize these pseudo operations, so they are included for completeness.

`.type expr`

The `.type` pseudo op is used within a `.def-.endef` pair. It gives `name` the C compiler type representation `expr`.

`.val expr`

The `.val` pseudo op is used with a `.def-.endef` pair. It gives `name` (in the `.def`) the value of `expr`. The type of `expr` determines the section for `name`.

`.tag str`

The `.tag` pseudo op is used in conjunction with a previously defined `.def` pseudo op. If the `name` of a `.def` is a structure or a union, `str` should be the `name` of that structure or union tag defined in a previous `.def-.endef` pair.

`.size expr`

The `.size` pseudo op is used with the `.def` pseudo op. If the `name` of a `.def` is an object such as a structure or an array, this gives it a total size of `expr.` `expr` must be a positive integer.

`.scl expr`

The `.scl` pseudo op is used with the `.def` pseudo op. Within the `.def` it gives `name` the storage class of `expr`. The type of `expr` should be positive.

`.line expr`

The `.line` pseudo op is used with the `.def` pseudo op. It defines the source line number of the definition of symbol `name` in the `.def.` `expr` should yield a positive value.

`.ln line [,addr]`

This pseudo op provides the relative source line number to the beginning of a function. It is used to pass information through to `sdb`.

`.file name`

The `.file` pseudo op is the source file name. Only one is allowed per source file. This **must** be the first line in an assembly file.

`.endef`

The `.endef` pseudo op is the ending bracket for a `.def`.

`.def name`

The `.def` pseudo op starts a symbolic description for symbol name. See `endef` (above). `name` is a symbol name.

`.dim expr [,expr]`

The `.dim` pseudo op is used with the `.def` pseudo op. If the `name` of a `.def` is an array, the expressions give the dimensions; up to four dimensions are accepted. The type of each expression should be positive.

dbx Pseudo Operations

`.stabs` *name type* `0` *desc value*

`.stabn` *type* `0` *desc value*

The `.stabs` and `.stabn` pseudo ops are debugger directives generated by the C compiler when the `-g` or `-go` options are used. *name* provides the symbol table name and type structure. *type* identifies the type of symbolic information (i.e., source file, global symbol, or source line). *desc* specifies the number of bytes occupied by a variable or type, or the nesting level for a scope symbol. *value* specifies an address or an offset.

B.4. Machine Instructions

This section describes the instructions that the assembler accepts. The detailed specification of how the particular instructions operate is not included; for this, see the Intel documentation (*80386 Programmer's Reference Manual*).

Differences between the SunOS and Intel 80386 Assemblers

The following list delineates the three main differences between the SunOS and Intel 80386 assembly languages. This explanation covers all aspects of translation from the Intel to SunOS assembler. On the SunOS assembler:

- All register names use the percent sign (`%`) as a prefix to distinguish them from symbol names.
- Instructions with two operands use the left one as the source and the right one as the destination. This follows the SunOS system's assembler convention, and **is reversed from Intel's notation**.
- Most instructions that can operate on a byte, word, or long may have `b`, `w`, or `l` appended to them. When an opcode is specified with no type suffix, it usually defaults to `long`. In general, the SunOS assembler derives its type information from the opcode, whereas the Intel assembler can derive its type information from the operand types. Where the type information is derived motivates the `b`, `w`, and `l` suffixes used in the SunOS assembler. For example, in the instruction `movw $1,%eax` the `w` suffix indicates the operand is a word.

Operands

Three kinds of operands are generally available to the instructions: register, memory, and immediate. Full descriptions of each type appear in Notational Conventions on page 180. Indirect operands are available **only** to jump and call instructions.

The assembler always assumes it is generating code for a 32-bit segment. When 16-bit data is called for (e.g., `movw %ax, %bx`), the assembler automatically generates the 16-bit data prefix byte.

Byte, word, and long registers are available on the 80386 processor. The code segment (`%cs`), instruction pointer (`%eip`), and flag register are not available as explicit operands to the instructions.

The names of the byte, word, and long registers available as operands and a brief description of each appear on the next page; the segment registers are listed also.

8-Bit (byte) General Registers

`%al`	Low byte of `%ax` register
`%ah`	High byte of `%ax` register
`%cl`	Low byte of `%cx` register
`%ch`	High byte of `%cx` register
`%dl`	Low byte of `%dx` register
`%dh`	High byte of `%dx` register
`%bl`	Low byte of `%bx` register
`%bh`	High byte of `%bx` register

16-Bit (word) General Registers

`%ax`	Low 16-bits of `%eax` register
`%cx`	Low 16-bits of `%ecx` register
`%dx`	Low 16-bits of `%edx` register
`%bx`	Low 16-bits of `%ebx` register
`%sp`	Low 16-bits of the stack pointer
`%bp`	Low 16-bits of the frame pointer
`%si`	Low 16-bits of the source index register
`%di`	Low 16-bits of the destination index register

32-Bit (long) General Registers

`%eax`	32-bit general register
`%ecx`	32-bit general register
`%edx`	32-bit general register
`%ebx`	32-bit general register
`%esp`	32-bit stack pointer
`%ebp`	32-bit frame pointer
`%esi`	32-bit source index register
`%edi`	32-bit destination index register

Segment Registers

`%cs`	Code segment register; all references to the instruction space use this register
`%ds`	Data segment register, the default segment register for most references to memory operands
`%ss`	Stack segment register, the default segment register for memory operands in the stack (i.e., default segment register for `%bp`, `%sp`, `%esp`, and `%ebp`)
`%es`	General-purpose segment register; some string instructions use this extra segment as their default segment
`%fs`	General-purpose segment register
`%gs`	General-purpose segment register

Introduction to Instruction Descriptions

This section describes the SunOS 386 instruction syntax. Refer to page 178 for the differences between the SunOS 386 and the Intel 386 assemblers.

Because the assembler assumes it is generating code for a 32-bit segment, it also assumes a 32-bit address and automatically precedes word operations with a 16-bit data prefix byte.

Notational Conventions

This section uses the following notational conventions:

- The mnemonics are expressed in a regular expression-type syntax. Alternatives separated by a vertical bar (|) and enclosed within square brackets ([ ]) denote that you must choose one of them. Alternatives enclosed within curly braces ({ }) denote that you can use one or none of them. The vertical bar separates different suffixes for operators or operands. For example, `imm[8|16|32]` indicates that an 8-, 16-, or 32-bit immediate value is permitted in an instruction.
- `imm[8|16|32|48]` — an immediate value. You define immediate values using the regular expression syntax previously described (see also Expressions and Immediate Values on page 182). If there is a choice between operand sizes, the assembler will choose the smallest representation.
- `reg[8|16|32]` — a general-purpose register, where each number indicates one of the following:
  - 32:`%eax`, `%ecx`, `%edx`, `%ebx`, `%esi`, `%edi`, `%ebp`, `%esp`
  - 16: `%ax`, `%cx`, `%dx`, `%bx`, `%si`, `%di`, `%bp`, `%sp`
  - 8: `%al`, `%ah`, `%cl`, `%ch`, `%dl`, `%dh`, `%bl`, `%bh`
- `mem[8|16|32|48]` — a memory operand; the 8, 16, 32, and 48 suffixes represent byte, word, doubleword, and inter-segment memory address quantities, respectively.
- `r/m[8|16|32]` — a general purpose register or memory operand; the operand type is determined from the suffix. They are: 8 = byte, 16 = word, and 32 = doubleword. The registers for each operand size are the same as `reg[8|16|32]` above.
- `creg` — a control register; the control registers are: `%cr0`, `%cr2`, or `%cr3`.
- `dreg` — a debug register; the debug registers are: `%db0`, `%db1`, `%db2`, `%db3`, `%db6`, `%db7`.
- `sreg` — a segment register; the segment registers are: `%cs`, `%ds`, `%ss`, `%es`, `%fs`, and `%gs`.
- `treg` — a test register; the test registers are: `%tr6` and `%tr7`.
- `cc` — condition codes; the 30 condition codes are:

a	above
ae	above or equal
b	below
be	below or equal
c	carry
e	equal
g	greater

ge	greater than or equal to
l	less than
le	less than or equal to
na	not above
nae	not above or equal to
nb	not below
nbe	not below or equal to
nc	no carry
ne	not equal
ng	not greater than
nge	not greater than or equal to
nl	not less than
nle	not less than or equal to
no	not overflow
np	not parity
ns	not sign
nz	not zero
o	overflow
p	parity
pe	parity even
po	parity odd
s	sign
z	zero

- disp[8|32] — the number of bits used to define the distance of a relative jump; because the assembler only supports a 32-bit address space, only 8-bit sign extended and 32-bit addresses are supported.
- immPtr — an immediate pointer; when the immediate form of a long call or a long jump is used, the selector and offset are encoded as an immediate pointer.

Addressing Modes

Addressing modes are represented by [sreg:][offset][([base][,index][,scale])], where all the items in the square brackets are optional, but at least one is necessary. If you use any of the items inside the parentheses, the parentheses are mandatory.

sreg is a segment register override prefix. It may be any segment register. If a segment override prefix is present, you must follow it by a colon before the offset component of the address. sreg does not represent an address by itself. An address must contain an offset component.

offset is a displacement from a segment base. It may be absolute or relocatable. A label is an example of a relocatable offset. A number is an example of an absolute offset.

base and index can be any 32-bit register. scale is a multiplication factor for the index register field. Refer to Intel's *80386 Programmer's Reference Manual* for more details on 80386 addressing modes.

Following are some examples of addresses:

`movl var, %eax`
Move the contents of memory location `var` into `%eax`.

`movl %cs:var, %eax`
Move the contents of the memory location `var` in the code segment into `%eax`.

`movl $var, %eax`
Move the address of `var` into `%eax`.

`movl array_base(%esi), %eax`
Add the address of memory location `array_base` to the contents of `%esi` to get an address in memory. Move the contents of this address into `%eax`.

`movl (%ebx, %esi, 4), %eax`
Multiply the contents of `%esi` by 4 and add this to the contents of `%ebx` to produce a memory reference. Move the contents of this memory location into `%eax`.

`movl struct_base(%ebx, %esi, 4), %eax`
Multiply the contents of `%esi` by 4, add this to the contents of `%ebx`, and add this to the address of `struct_base` to produce an address. Move the contents of this address into `%eax`.

Expressions and Immediate Values

An immediate value is an expression preceded by a dollar sign:

```
immediate: "$" expr
```

Immediate values carry the absolute or relocatable attributes of their expression component. Immediate values cannot be used in an expression, and should be considered as another form of address, i.e., the immediate form of address.

Processor Extension Instructions

The following five sections list instructions that are new or extended (to 32-bit operands) in the 80386 compared with the 80286 microprocessor.

Control and Test Register Instructions

```
mov{l}     creg, reg32
mov{l}     dreg, reg32
mov{l}     reg32, creg
mov{l}     reg32, dreg
mov{l}     treg, reg32
mov{l}     reg32, treg
```

NOTE *The UNIX assembler accepts* `mov` *or* `movl` *as exactly the same instruction for the control and test register group.*

New Condition Code Instructions

```
jcc        disp32
setcc      r/m8
```

New Bit Instructions

All the new bit instructions are only defined for word and long register or memory operands.

```
bt{wl}      reg[16|32], r/m[16|32]
bt{wl}      imm8, r/m[16|32]
bts{wl}     imm8, r/m[16|32]
bts{wl}     reg[16|32], r/m[16|32]
btr{wl}     imm8, r/m[16|32]
btr{wl }    reg[16|32], r/m[16|32]
btc{wl}     imm8, r/m[16|32]
btc{wl}     reg[16|32], r/m[16|32]
bsf{wl}     reg[16|32], r/m[16|32]
bsr{wl}     reg[16|32], r/m[16|32]
shld{wl}    imm8, reg[16|32], r/m[16|32]
shld{wl}    reg[16|32], r/m[16|32]
shrd{wl}    imm8, reg[16|32], r/m[16|32]
shrd{wl}    reg[16|32], r/m[16|32]
```

NOTE *All the bit operation mnemonics without a type suffix default to* `long`.

New Arithmetic Instruction

```
imul        r/m[16|32], reg[16|32]
```

NOTE *This is the uncharacterized multiply. It has a 16- or 32-bit product, as opposed to a 32- or 64-bit product.*

New Move with Zero or Sign Extension Instructions

```
movzbw      r/m8, reg16
movzbl      r/m8, reg32
movzwl      r/m16, reg32
movsbw      r/m8, reg16
movsbl      r/m8, reg32
movswl      r/m16, reg32
```

New Data Movement Instructions

```
clr{bwl}    r/m[8|16|32]
lea{wl}     mem32, reg[16|32]
mov{bwl}    r/m[8|16|32], reg[8|16|32]
mov{bwl}    reg[8|16|32], r/m[8|16|32]
mov{bwl}    imm[8|16|32], r/m[8|16|32]
pop{wl}     r/m[16|32]
popa{wl}
push{bwl}   imm[8|16|32]
push{wl}    r/m[16|32]
pusha{wl}
xchg{bwl}   reg[8|16|32], r/m[8|16|32]
```

NOTE 1 `pushb` *sign extends the immediate byte to a* `long`, *and pushes a* `long` *(4 bytes) onto the stack.*

NOTE 2 *When a type suffix is not used with a data movement mnemonic, the type defaults to* `long`. *The SunOS assembler does not derive the type of the operands from the operands. (See the last bullet item in Differences between the SunOS and Intel 80386 Assemblers on page 178.)*

Segment Register Instructions

```
lds{wl}     mem[32|48],  reg[16|32]
les{wl}     mem[32|48],  reg[16|32]
lfs{wl}     mem[32|48],  reg[16|32]
lgs{wl}     mem[32|48],  reg[16|32]
lss{wl}     mem[32|48],  reg[16|32]
movw        sreg[cs|ds|ss|es] , r/m16
movw        r/m16, sreg[cs|ds|ss|es]
popw        sreg[ds|ss|es|fs|gs]
pushw       sreg[cs|ds|ss|es|fs|gs]
```

NOTE 1 *The* `pushw` *and* `popw` *push and pop 16-bit quantities. This is done by using a data size override byte (OSP byte).*

NOTE 2 *When the type suffix is not used with the* `lds`, `les`, `lfs`, `lgs`, *and* `lss` *instructions, a 48-bit pointer is assumed.*

NOTE 3 *Because the assembler assumes no type suffix means a type of* `long`, *the type suffix of* `w`, *when working with the segment registers, is mandatory.*

I/O Instructions

```
in{bwl}     imm8
in{bwl}     %dx
ins{bwl}    %dx
out{bwl}    imm8
out{bwl}    %dx
outs{bwl}   %dx
```

NOTE *When the type suffix is left off the I/O instructions, they default to* `long`. *Therefore,* `in` = `inl`, `out` = `outl`, `ins` = `insl`, *and* `outs` = `outsl`.

Flag Instructions

```
lahf
sahf
popf{wl}
pushf{wl}
cmc
clc
stc
cli
sti
cld
std
```

NOTE *When the type suffix is not used, the* `pushf` *and* `popf` *instructions default to* `long`, `pushf` = `pushfl` *and* `popf` = `popfl`. *A* `pushw` *or* `popw` *will push or pop a 16-bit quantity. This is done by using the OSP prefix byte.*

Arithmetic/Logical Instructions

```
add{bwl}    reg[8|16|32], r/m[8|16|32]
add{bwl}    r/m[8|16|32], reg[8|16|32]
add{bwl}    imm[8|16|32], r/m[8|16|32]
adc{bwl}    reg[8|16|32], r/m[8|16|32]
adc{bwl}    r/m[8|16|32], reg[8|16|32]
adc{bwl}    imm[8|16|32], r/m[8|16|32]
sub{bwl}    reg[8|16|32], r/m[8|16|32]
sub{bwl}    r/m[8|16|32], reg[8|16|32]
sub{bwl}    imm[8|16|32], r/m[8|16|32]
sbb{bwl}    reg[8|16|32], r/m[8|16|32]
sbb{bwl}    r/m[8|16|32], reg[8|16|32]
sbb{bwl}    imm[8|16|32], r/m[8|16|32]
cmp{bwl}    reg[8|16|32], r/m[8|16|32]
cmp{bwl}    r/m[8|16|32], reg[8|16|32]
cmp{bwl}    imm[8|16|32], r/m[8|16|32]
inc{bwl}    r/m[8|16|32]
dec{bwl}    r/m[8|16|32]
test{bwl}   reg[8|16|32], r/m[8|16|32]
test{bwl}   r/m[8|16|32], reg[8|16|32]
test{bwl}   imm[8|16|32], r/m[8|16|32]
sal{bwl}    imm8, r/m[8|16|32]
sal{bwl}    %cl, r/m[8|16|32]
shl{bwl}    imm8, r/m[8|16|32]
shl{bwl}    %cl, r/m[8|16|32]
sar{bwl}    imm8, r/m[8|16|32]
sar{bwl}    %cl, r/m[8|16|32]
shr{bwl}    imm8, r/m[8|16|32]
shr{bwl}    %cl, r/m[8|16|32]
not{bwl}    r/m[8|16|32]
neg{bwl}    r/m[8|16|32]
bound{wl}   reg[16|32], r/m[16|32]
and{bwl}    reg[8|16|32], r/m[8|16|32]
and{bwl}    r/m[8|16|32], reg[8|16|32]
and{bwl}    imm[8|16|32], r/m[8|16|32]
or{bwl}     reg[8|16|32], r/m[8|16|32]
or{bwl}     r/m[8|16|32], reg[8|16|32]
or{bwl}     imm[8|16|32], r/m[8|16|32]
xor{bwl}    reg[8|16|32], r/m[8|16|32]
xor{bwl}    r/m[8|16|32], reg[8|16|32]
xor{bwl}    imm[8|16|32], r/m[8|16|32]
```

NOTE *When the type suffix is not included in an arithmetic or logical instruction, it defaults to a* `long`.

Multiply and Divide Instructions

`imul{wl}`	`imm[16|32], r/m[16|32], reg[16|32]`
`mul{bwl}`	`r/m[8|16|32]`
`div{bwl}`	`r/m[8|16|32]`
`idiv{bwl}`	`r/m[8|16|32]`

NOTE *When the type suffix is not included in a multiply or divide instruction, it defaults to a* `long`.

Conversion Instructions

`cbtw`
`cwtd`
`cwtl`
`cltd`

NOTE *Convert byte to word:* `%al` -> `%ax`
Convert word to double: `%ax` -> `%dx:%ax`
Convert word to long: `%ax` -> `%eax`
Convert long to double: `%eax` -> `%edx:%eax`

Decimal Arithmetic Instructions

`daa`
`das`
`aaa`
`aa`
`aam`
`aad`

Coprocessor Instructions

`wait`
`esc`

String Instructions

`movs[bwl]`	
`movs`	same as `movsl`
`smov[bwl]`	same as `movs[bwl]`
`smov`	same as `smovl`
`cmps[bwl]`	
`cmps`	same as `cmpsl`
`scmp[bwl]`	same as `cmps[bwl]`
`scmp`	same as `scmpl`
`stos[bwl]`	
`stos`	same as `stosl`
`ssto[bwl]`	same as `stos[bwl]`
`ssto`	same as `sstol`
`lods[bwl]`	
`lods`	same as `lodsl`
`slod[bwl]`	same as `lods[bwl]`
`slod`	same as `slodl`
`scas[bwl]`	
`scas`	same as `scasl`

`ssca[bwl]`	same as `scas[bwl]`
`ssca`	same as `sscal`
`xlat`	
`rep`	
`repnz`	
`repz`	

NOTE *All Intel string op mnemonics default to* long.

Procedure Call and Return Instructions

`lcall`	`immPtr`
`lcall`	`r/m48` (indirect)
`lret`	
`lret`	`imm16`
`call`	`disp32`
`call`	`r/m32` (indirect)
`ret`	
`ret`	`imm16`
`enter`	`imm16, imm8`
`leave`	

Jump Instructions

`jcc`	`disp[8|32]`
`jcxz`	`disp[8|32]`
`loop`	`disp[8|32]`
`loopnz`	`disp[8|32]`
`loopz`	`disp[8|32]`
`jmp`	`disp[8|32]`
`ljmp`	`immPtr`
`jmp`	`r/m32` (indirect)
`ljmp`	`r/m48` (indirect)

NOTE *The Sun386i assembler optimizes for SDIs (Span Dependent Instructions). Consequently, intra-segment jumps are optimized to their short forms when possible.*

Interrupt Instructions

`int3`	
`int`	`imm8`
`into`	
`iret`	

Protection Model Instructions

`sldt`	`r/m16`
`str`	`r/m16`
`lldt`	`r/m16`
`ltr`	`r/m16`
`verr`	`r/m16`
`verw`	`r/m16`
`sgdt`	`r/m32`
`sidt`	`r/m32`

```
lgdt      r/m32
lidt      r/m32
smsw      r/m32
lmsw      r/m32
lar       r/m32, reg32
lsl       r/m32, reg32
clts
```

Miscellaneous Instructions

```
lock
nop
hlt
addr16
data16
```

B.5. Translation Tables for SunOS to Intel Float Mnemonics

The following tables show the relationship between the SunOS and Intel mnemonics. The mnemonics are organized into the same functional categories as the Intel mnemonics. (The Intel mnemonics appear in the second section of the 80287 numeric supplement.)

The notational conventions used in the table are as follows:

- When letters appear within square brackets ([ ]) exactly one of the letters is required.
- If letters appear within curly braces ({ }) then either one or none of the letters is required.
- When a group of letters is separated from other letters by a bar (|) within square brackets or curly braces, then the group of letters between the bars or between a bar and a closing bracket or brace is considered an atomic unit.

For example, `fld[lst]` means `fldl`, `flds`, or `fldt`.; `fst{ls}` means `fst`, `fstl`, or `fsts`; and `fild{l|ll}` means `fild`, `fildl`, or `fildll`.

The SunOS operators are built from the Intel operators by adding suffixes to them. The 80287/387 deals with three data types: integer, packed decimal, and real. The SunOS assembler is not typed; the operator has to carry with it the type of data item it is operating on. If the operation is on an integer, the following suffixes apply: `l` for Intel's `short` (32 bits), and `ll` for Intel's `long` (64 bits). If the operator applies to reals, then: `s` is `short` (32 bits), `l` is `long` (64 bits), and `t` is temporary real (80 bits).

Real Transfers

SunOS	*INTEL*	*Operation*
`fld[lst]`	`fld`	Load real
`fst{ls}`	`fst`	Store real
`fstp{lst}`	`fstp`	Store real and pop
`fxch`	`fxch`	Exchange registers

Integer Transfers

SunOS	*INTEL*	*Operation*
`fild{l|ll}`	`fild`	Integer load
`fist{l}`	`fist`	Integer store
`fistp{l|ll}`	`fistp`	Integer store and pop

Packed Decimal Transfers

SunOS	*INTEL*	*Operation*
`fbld`	`fbld`	Packed decimal (BCD) load
`fbstp`	`fbstp`	Packed decimal (BCD) store and pop

Addition

SunOS	*INTEL*	*Operation*
`fadd{ls}`	`fadd`	Real add
`faddp`	`faddp`	Real add and pop
`fiadd{l}`	`fiadd`	Integer add

Subtraction

SunOS	*INTEL*	*Operation*
`fsub{ls}`	`fsub`	Subtract real
`fsubp`	`fsubp`	Subtract real and pop
`fsubr{ls}`	`fsubr`	Subtract real reversed
`fsubrp`	`fsubrp`	Subtract real reversed and pop
`fisub{l}`	`fisub`	Integer subtract
`fisubr{l}`	`fisubr`	Integer subtract reverse

Multiplication

SunOS	*INTEL*	*Operation*
`fmul{ls}`	`fmul`	Multiply real
`fmulp`	`fmulp`	Multiply real and pop
`fimul{l}`	`fimul`	Integer multiply

Division

SunOS	*INTEL*	*Operation*
`fdiv{ls}`	`fdiv`	Divide real
`fdivp`	`fdivp`	Divide real and pop
`fdivr{ls}`	`fdivr`	Divide real reversed
`fdivrp`	`fdivrp`	Divide real reversed and pop
`fidiv{l}`	`fidiv`	Integer divide
`fidivr{l}`	`fidivr`	Integer divide reversed

Other Arithmetic Operations

SunOS	*INTEL*	*Operation*
`fsqrt`	`fsqrt`	Square root
`fscale`	`fscale`	Scale
`fprem`	`fprem`	Partial remainder
`frndint`	`frndint`	Round to integer
`fxtract`	`fxtract`	Extract exponent and significand
`fabs`	`fabs`	Absolute value
`fchs`	`fchs`	Change sign

Comparison Instructions

SunOS	*INTEL*	*Operation*
`fcom{ls}`	`fcom`	Compare real
`fcomp{ls}`	`fcomp`	Compare real and pop
`fcompp`	`fcompp`	Compare real and pop twice
`ficom{l}`	`ficom`	Integer compare
`ficomp{l}`	`ficomp`	Integer compare and pop
`ftst`	`ftst`	Test
`fxam`	`fxam`	Examine

Transcendental Instructions

SunOS	*INTEL*	*Operation*
`fptan`	`fptan`	Partial tangent
`fpatan`	`fpatan`	Partial arctangent
`f2xm1`	`f2xm1`	2^{x-1}
`fyl2x`	`fyl2x`	$Y * \log_2 X$
`fyl2xp1`	`fyl2xp1`	$Y * \log_2 (X+1)$

Constant Instructions

SunOS	*INTEL*	*Operation*
`fldl2e`	`fldl2e`	Load $\log_e$ E
`fldl2t`	`fldl2t`	Load $\log_2$ 10
`fldlg2`	`fldlg2`	Load $\log_2$ 2
`fldln2`	`fldln2`	Load $\log_e$ 2
`fldpi`	`fldpi`	Load pi
`fldz`	`fldz`	Load + 0

Processor Control Instructions

SunOS	*INTEL*	*Operation*
`finit/fnint`	`finit/fnint`	Initialize processor
`fnop`	`fnop`	No operation
`fsave/fnsave`	`fsave/fnsave`	Save state
`fstcw/fnstcw`	`fstcw/fnstcw`	Store control word
`fstenv/fnstenv`	`fstenv/fnstenv`	Store environment
`fstsw/fnstsw`	`fstsw/fnstsw`	Store status word
`frstor`	`frstor`	Restore state
`fsetpm`	`fsetpm`	Set protected mode
`fwait`	`fwait`	CPU wait
`fclex/fnclex`	`fclex/fnclex`	Clear exceptions
`fdecstp`	`fdecstp`	Decrement stack pointer
`ffree`	`ffree`	Free registers
`fincstp`	`fincstp`	Increment stack pointer

C

File System Layout

C

File System Layout

This appendix describes the file system layout for the Sun386i system. Much of this structure is similar to that of the SunOS 4.0 system on other architectures. The 4.0 layout is intended to make it easier for a single server to support clients of different architectures. The Sun386i layout also provides a standard place to mount additional disks and makes it easier for users to access network files and applications without knowing the location of files and without changing `.login` or `.cshrc` files.

Some file system differences also exist between Sun386i systems and other Sun systems to accommodate the division of Sun386i system software. The Sun386i system groups much of system software into related files and programs called clusters, which users can add to their systems (Chapter 9 contains details). For compatibility between systems, and because previously existing programs expect to find files in their traditional directories, some directories now contain symbolic links to directories that contain the actual files.

If you'll be distributing software for the Sun386i system, the `/usr/local` directory is important to you. Page 201 describes this directory, and pages 206–207 describe the preferred directory structure for releasing software for this workstation.

C.1. Terms

The Sun Network File System (NFS) allows any computer with a local disk to act as a file server by exporting its file systems to clients on a network. The client computers may themselves be file servers of other file systems.

The Yellow Pages (YP) is a distributed network database. Key information about the systems and users on the network is stored in the YP database on the master server and slave servers. The master server keeps the master copy of the database, using it to update slave servers. The YP is stored on the master server and all the slave servers to ensure the availability of the database in case a server goes down. However, the master server must be running for the updates to YP to occur. *Sun386i Advanced Administration* discusses NFS and YP in more detail.

The automounter (`automount`(8)) is a daemon that automatically and transparently mounts an NFS file system at a temporary mount point whenever a file or directory within that system is opened. The mounted file system is made available using a symbolic link to the mount point. *Sun386i Advanced Administration* and the `automount`(8) `man` page contain more information.

C.2. Layout Overview

The Sun386i file system layout:

- Provides easier maintenance of servers and clients
- Enables easier mixing of remote and local file systems
- Provides cleaner support of multiple architectures
- Provides a hierarchy that accounts for growth, enabling users to mount additional disks without affecting file names
- Minimizes disruption to existing programs when files are moved
- Minimizes symbolic link confusion
- Minimizes user confusion when they log in to different systems

Sun386i software is divided among three primary file systems: / (root), /usr, and /files. Each is described briefly in the next section.

System Disk

The disk that the system boots from is called the *system disk*. The standard system disk layout consists of the following special device files and partitions:

- /dev/roota — contains the root (/) file system; 8 Mbytes
- /dev/rootb — contains the system's swap area; 16 Mbytes
- /dev/rootg — contains /usr file system; 19 Mbytes
- /dev/rooth — contains the /files file system; size depends on disk size

The default /etc/fstab file contains entries to mount the corresponding file systems using the partitions listed above. This enables users to boot systems from any disk without modifying /etc/fstab.

The standard Sun386i file systems and directories are briefly described below. Subsequent sections detail the contents of each one.

/ (root) File System

/ (root) is the major SunOS file system, located at the top of the hierarchical file system tree. It contains machine-specific files and directories crucial for system operation, such as the kernel, a device directory listing the equipment for the configuration, and programs used for booting the multiuser version of the operating system. The contents of / is described on the following page.

/usr File System

/usr contains executable commands, system programs, and library routines, as well as some executables that were formerly under / (such as system administration programs). This is a read-only file system to enable network-wide mounting and sharing. Because /usr is read-only, users see the same files regardless of where they log in. /usr is intentionally very full, so do not add to this file system. The contents of /usr is shown on page 200.

/files File System

/files contains free space remaining after allocation of the root, swap, and usr partitions. /files contains home directories, unbundled and third-party applications, optionally loaded Application Supplement and Developer's Toolkit clusters, and root, swap, and dump directories for diskless clients. The contents of /files is delineated on page 202. Disks added to the system are mounted on /files*n*, as described in the next section, Additional Disks.

/home Directory

/home is used in conjunction with the automounter (automount(8)) to provide transparent access to a user's home directory, regardless of where the directory resides on the network. /home is described further as part of the / file system on the next page.

/export Directory

/export contains symbolic links to the local files and directories that diskful systems export to other machines on the network. Only the links, not the directories themselves, are stored in /export. Other systems on the network cannot see the actual location of exported directories. Pages 203–205 provide more information on /export.

/vol Directory

/vol is an automount(8) directory for *volumes*. A volume is a collection of related files dedicated to the same function, such as all files required to run an unbundled or third-party application. Administrators can move volumes between partitions and systems with relative ease. /vol is described more fully on page 205.

Additional Disks

Additional disks added to the Sun386i system have one partition, sd0c, which provides access to the entire disk. Each additional disk is mounted on /files*n*, where *n* indicates the order of the disk added (/files1, /files2, and so on). *n* does not correspond to the disk's special device file name in /dev.

C.3. / File System

/ contains the following files and directories:

```
/bin         /files<n>      /mnt      /tftpboot   VERSION
boot.S386    /home          /net      /tmp        /vmunix
/dev         kadb           /sbin     /tmp_mnt    /vol
/etc         /lib           /stand    /usr
/export      /lost+found    /sys      /var
```

/bin

Starting with Release 4.0, /bin is a symbolic link to /usr/bin, described on page 200.

boot.S386

boot.S386 is the program used to load the SunOS operating system.

/dev

/dev is the device directory, which contains all device files (also called device nodes) such as rst0 (quarter-inch tape drive), /dev/ttya (serial port), or /dev/pp0 (parallel port), for a particular configuration.

/etc

/etc contains system-specific data files and subdirectories primarily used by system administrators; includes the files created during installation.

/export

/export is described in Section C.6, starting on page 203.

/files<*n*>

/files<*n*> is the mount point for the /files<*n*> file system, described in Section C.5 starting on page 202.

/home

/home is an automount(8) directory that provides automatic access to home directories for all users on the network. By default, users' home directories are stored in /files<*n*>/home/*groupname*/*username* on various systems, but passwd Yellow Pages (YP) entries for each user specify the home directory path as /home/*username*. The automounter (automount(8)) takes references to /home/*username* and uses the auto.home YP map to return symbolic links to the home or ~*username* directory. /home is shipped empty; /home/*username* does not exist as part of the file system on disk, but rather is created only after an automount reference is made to it.

If the auto.home entry indicates that the home directory is on a remote system, the automounter creates a temporary mount point under /tmp_mnt and uses this point to mount the remote directory onto the local system via NFS. The automounter returns a symbolic link to the mount point.

If the home directory is on the local system, the automounter returns a symbolic link to the directory. For more information on the automounter, see *Sun386i Advanced Administration*, and the auto.home(5) and automount(8) man pages.

kadb

kadb(8) is the kernel debugger program.

/lib

/lib is a symbolic link to /usr/lib, described on page 200.

/lost+found

/lost+found is usually empty. However, if / becomes damaged, the file system check program (fsck(8)) places links to any files that it cannot link elsewhere in this file system in /lost+found.

/mnt

/mnt is a mount point for temporarily mounting systems with the mount(8) command.

/net

/net is an automount(8) point used by the SNAP backup facility. Experienced users can also use /net to access directories on remote systems, although /home and /vol are preferred. Before using /net, make sure the system is an NFS file server (diskful system) and the path name is exported (/net/*hostname*/export/*pathname*).

/sbin

/sbin contains executable files necessary to check and mount the /usr file system and to bring up a multiuser system at boot time:

- /sbin/fsck — checks and repairs the file system
- /sbin/init — performs process control initialization
- /sbin/mount — mounts file systems
- /sbin/netconfig — configures the network

- /sbin/reboot — reboots the system
- /sbin/sh — standard command interpreter

/stand

/stand is the diagnostics directory for standalone programs. Contains a standalone copy program, tape boot program, and an extra copy of the boot program (boot.S386).

/sys

/sys is a symbolic link to /usr/sys, described on page 201.

/tftpboot

/tftpboot contains the files necessary to boot diskless clients on the network.

/tmp

/tmp holds files temporarily; utilities such as cc(1) and ar(1) create temporary data files in /tmp. All files in /tmp, with the exception of subdirectories, are deleted each time the system is rebooted.

/tmp_mnt

/tmp_mnt is the directory that the automounter (automount(8)) uses to make mount points for temporary file systems. Do not add files to or remove files from this directory.

/usr

/usr is the mount point for the /usr file system, described on the following page.

/var

/var contains the following directories and symbolic links. Diskless systems contain directories, not the links to directories located elsewhere.

- /var/adm — a symbolic link to /files<*n*>/var/adm
- /var/crash — a symbolic link to /files<*n*>/var/crash
- /var/log — directory for log files; shipped empty
- /var/preserve — a symbolic link to /files<*n*>/var/preserve
- /var/recover — directory for crash recovery scripts (on the Sun386i system only); shipped empty
- /var/spool — a symbolic link to /files<*n*>/var/spool
- /var/tmp — a symbolic link to /files<*n*>/var/tmp
- /var/yp — directory containing Yellow Pages databases

VERSION

VERSION is a text file specifying the version of the root file system.

/vmunix

/vmunix is the SunOS system kernel.

/vol

/vol is an automount(8) directory described in Section C.7 starting on page 205; shipped emtpy.

C.4. /usr File System

/usr is read-only and shared. It contains only architecture-specific executables and libraries, including the files and directories:

5bin
5include
5lib
adm
bin
boot
dict
etc
games
hosts
include
lib
local
lost+found
man
mdec
old
pub
sccs
share
spool
src
stand
sys
sysex
tmp
ucb
VERSION

/usr/5bin

/usr/5bin contains UNIX System V binary files.

/usr/5include

/usr/5include contains UNIX System V include files.

/usr/5lib

/usr/5lib contains UNIX System V libraries.

/usr/adm

/usr/adm is a symbolic link to /files<*n*>/var/adm.

/usr/bin

/usr/bin contains basic SunOS operating system commands, including those formerly located in /bin, such as ls(1V), cat(1V), and mkdir(1).

/usr/boot

/usr/boot is a directory that contains symbolic links to files in /sbin and /.

/usr/dict

/usr/dict is a database that contains English language spelling lists used by the spell(1) spelling checker, if the optional cluster spell_check is loaded; shipped empty.

/usr/etc

/usr/etc contains the commands and files used for system administration and maintenance.

/usr/games

/usr/games is a symbolic link to /files<*n*>/loaded/appl/games/games, which exists only if the optional games cluster is loaded.

/usr/hosts

/usr/hosts is a symbolic link to /files<*n*>/hosts.

/usr/include

/usr/include is set up to contain all of the standard include (header) files used in C programs; these files, traditionally named with a .h extension, contain definitions of useful constants and macros.

/usr/lib

/usr/lib contains more than 100 files used by SunOS utilities and files formerly located in /lib (which is now a symbolic link to /usr/lib).

/usr/lib includes the subdirectory /usr/lib/load, which contains the cluster size and file database used by the load(1), loadc(1), unload(1), unloadc(1), and cluster(1) commands; also contains the filesizes file which lists the names and sizes of files within each cluster (on the Sun386i system only).

/usr/local

/usr/local is reserved for third-party and unbundled products. On Sun386i systems, it is a symbolic link to /files<*n*>/local, described on the next page.

/usr/lost+found

/usr/lost+found is usually empty. However, if /usr becomes damaged, the file system check program (fsck(8)) places links to any files that it cannot link elsewhere in this file system in /usr/lost+found.

/usr/man

/usr/man is a symbolic link to /usr/share/man, described later in this section.

/usr/mdec

/usr/mdec is a symbolic link to /files<*n*>/loaded/appl/advanced_admin/mdec, which exists only if the optional advanced_admin cluster (containing boot blocks and the install boot program for the Sun386i system) is loaded.

/usr/old

/usr/old is a symbolic link to /files<*n*>/loaded/appl/old, which exists only if the optional old cluster is loaded. The old cluster contains commands that have been phased out but retained for compatibility.

/usr/pub

/usr/pub contains data files used in formatting and printing.

/usr/sccs

/usr/sccs is a symbolic link to /files<*n*>/loaded/devel/sccs/sccs, which exists only if the optional sccs cluster is loaded.

/usr/share

/usr/share contains architecture-independent sharable files, shown below:

- /usr/share/lib — contains tab set, termcap, time zone, and terminal information
- /usr/share/man — symbolic link to /files<*n*>/loaded/appl/man_pages

/usr/spool

/usr/spool is a symbolic link to /files<*n*>/var/spool.

/usr/src

/usr/src is a symbolic link to /files/src, described on page 203.

/usr/stand

/usr/stand is a symbolic link to /stand, described on page 199.

/usr/sys

/usr/sys is a symbolic link to /files<*n*>/loaded/devel/config/sys, which exists only if the optional config cluster is loaded.

/usr/sysex

/usr/sysex contains System Exerciser files.

/usr/tmp

/usr/tmp is a symbolic link to /files<*n*>/var/tmp.

/usr/ucb

/usr/ucb contains commands that originated with the Berkeley UNIX system (ucb is an acronym for University of California at Berkeley).

/usr/VERSION

/usr/VERSION is a text file specifying the version of this /usr file system.

C.5. /files<*n*> File System

/files<*n*> could contain the directories shown in this section (not all of these will exist on disks added to the expansion unit). /files is the name of this file system on the system disk. The name of this file system on the first additional disk added to the system is /files1, for the second disk added /files2, and so on.

```
dump        hosts       local.unix    src
exec        loaded      lost+found    swap
home        local       root          var
```

/files<*n*>/dump

/files<*n*>/dump is reserved for kernel core dumps of diskless clients if they crash; shipped empty.

/files<*n*>/exec

/files<*n*>/exec contains the native executables (typically symbolic links to /usr file systems) of each Sun workstation architecture and each SunOS release on the server's disk.

/files<*n*>/home

/files<*n*>/home is reserved for the home directories of each user on a server (/files<*n*>/home/*groupname*/*username*).

/files<*n*>/hosts

/files<*n*>/hosts contains the MAKEHOSTS script, which creates a symbolic link to the rsh(1C) command for each host in the /etc/hosts file. Use of MAKEHOSTS is not recommended because it does not work in heterogeneous networks; included for compatibility.

/files<*n*>/loaded

/files<*n*>/loaded is reserved for optional software clusters added to the Sun386i system. When loaded, clusters reside in or beneath the two directories shown below. Users can execute commands within clusters by just entering the command name, which is a link to the command location in one of these directories.

- appl — mount or load points for Application SunOS optional clusters; shipped empty.
- devel — mount or load points for Developer's Toolkit clusters; shipped empty.

/files<*n*>/local

/files<*n*>/local is reserved for third-party and unbundled software added to a Sun386i system; shipped empty. The suggested location for both third-party and Sun unbundled applications is /usr/local/*application_name*, which on Sun386i systems, is a link to /files/local/*application_name*. For more information on the subdirectories you should include with your applications, refer to Section C.8 on pages 206–207.

To enable network-wide access of files and applications, administrators can include links to applications in /files<*n*>/local.unix, as described below.

/files<*n*>/local.unix

/files<*n*>/local.unix is on the server that contains /vol/local (the Yellow Pages master by default). It contains:

- bin — contains shell scripts; also designed to contain links that point to /usr/bin/start_applic for each application that has its own volume. System administrators can create links in

/files/local.unix/bin that have the same name as the application. Then, if an administrator exports the application and makes it accessible from a volume, any user on the network can execute the application simply by entering *application_name*. The start_applic script sets environment variables for the application, and begins application execution on the user's machine.

- bin.*arch* — contains architecture-dependent files, where *arch* can be either sun1, sun2, sun3, sun4, or sun386.

Sections C.6 and C.7 briefly describe exporting and volumes, and pages 96–98 list the steps to export and create a volume for an application; *Sun386i Advanced Administration* contains details.

/files<*n*>/lost+found

/files<*n*>/lost+found is usually empty. However, if /files<*n*> becomes damaged, the file system check program (fsck(8)) places links to any files that it cannot link elsewhere in this file system in /files<*n*>/lost+found.

/files<*n*>/root

/files<*n*>/root is reserved for root directories for all diskless clients of a server (/files<*n*>/root/*hostname*).

/files<*n*>/src

/files<*n*>/src is not present on all configurations. If your machine is licensed to contain source code, it will reside in this directory.

/files<*n*>/swap

/files<*n*>/swap is reserved for individual swap areas for all diskless clients of a server (/files<*n*>/swap/*hostname*).

/files<*n*>/var

/files<*n*>/var contains subdirectories that have files that tend to grow, such as:

- /files<*n*>/var/adm — contains system accounting and log files
- /files<*n*>/var/crash — is reserved for kernel core dumps of servers and standalone systems if they crash; shipped empty.
- /files<*n*>/var/preserve — holds files saved by the vi and ex editors if the system crashes
- /files<*n*>/var/spool — contains files used for printing and other spooling functions
- /files<*n*>/var/sysex — directory where the System Exerciser writes its temporary and log files. The System Exerciser runs under the SunOS system and verifies operation of the total system, including operating system software (on Sun386i systems only).
- /files<*n*>/var/tmp — contains temporary files placed here by programs; unlike /tmp, the files in this directory are not deleted when you reboot the system

C.6. /export Directory

/export contains symbolic links to local directories that diskful systems export to other machines on the network. Many of these links already exist on the system, and others are created when performing certain administrative functions via SNAP or New User Accounts. In both cases, the system edits the /etc/exports file to include information on exported directories.

To export additional directories:

1. Create a symbolic link to the directory; for example, /export/local/*application_name* is the recommended symbolic link to /usr***<n>***/local/*application_name*.
2. Include that link in /etc/exports.
3. Run exportfs -a to notifiy the mount daemon of the change.

Sun386i Advanced Administration and the exports(5) and exportfs(8) man pages contain more information about exporting directories.

The leaf nodes (final components of path names) that the system typically exports are shown below.

/export/dump

If this system is a boot server for diskless clients, /export/dump contains symbolic links to a client system's dump directory if the server is saving crash dumps for clients:

> /export/dump/*client_systemname* is a symbolic link to
> /files***<n>***/dump/*client_systemname*

/export/exec

/export/exec contains a symbolic link for each software architecture of the SunOS system loaded on this workstation. Each link points to the particular release's location:

> /export/exec/*arch* is a symbolic link to /export/exec/*arch*. *<OS release>*, which is a symbolic link to /usr if the server is running the same *<arch>*.*<OS release>* as the /exec being exported; if this is not the case, then /export/exec/*<arch>*.*<OS release>* is a symbolic link to /files***<n>***/exec/*<arch>*.*<OS release>*

where *<arch>* can be either sun1, sun2, sun3, sun4, or sun386 and *<OS release>* has the format SunOS4.0, SunOS4.1BETA1, SunOS4.1BETA2, and so on.

Diskless clients mount /usr from their boot server, using the entry that Automatic System Installation places in /etc/bootparams:

> usr = *server*:/export/exec/*<arch>*.*<OS release>*

/export/home

If this system has a disk with users' home directories, /export/home contains symbolic links to each user's home directory:

> /export/home/*groupname*/*username* is a symbolic link to
> /files***<n>***/home/*groupname*/*username*

/export/loaded

/export/loaded contains a symbolic link that points to each optional Application Supplement or Developer's Toolkit cluster added to this system.

If the architectures of both machines are the same, then:

> /export/loaded/*<arch>*.*<OS release>* is a symbolic link to
> /files***<n>***/loaded

For different architectures:

> `/export/loaded/`*<arch>.<OS release>* is a symbolic link to
> `/files`*<n>*`/loaded/`*<arch>.<OS release>*

where *<arch>* can be either `sun1`, `sun2`, `sun3`, `sun4`, or `sun386` and *<OS release>* has the format `SunOS4.0`, `SunOS4.1BETA1`, `SunOS4.1BETA2`, and so on.

Diskless clients mount `/files`*<n>*`/loaded` from their boot server, using the entry that Automatic System Installation places in `/etc/bootparams`:

> `/files`*<n>*`/loaded -` *server*`:/export/loaded`

`/export/local`

`/export/local` is reserved for symbolic links to exported applications, in the format `/export/local/`*application_name*. Links resolve to `/usr/local/`*application_name*.

`/export/local.unix`

`/export/local.unix` is a symbolic link to `/files`*<n>*`/local.unix`, described on pages 202–203.

`/export/root`

If this system is a boot server for diskless clients, `/export/root` contains symbolic links to each client system's root directory:

> `/export/root/`*client_systemname* is a symbolic link to
> `/files`*<n>*`/root/`*client_systemname*

Diskless clients mount / from their boot server, using the entry that Automatic System Installation places in `/etc/bootparams`:

> `root =` *server*`:/export/root/`*client_systemname*

`/export/swap`

If this system is a boot server for diskless clients, `/export/swap` contains symbolic links to each client system's swap file:

> `/export/swap/`*client_systemname* is a symbolic link to
> `/files`*<n>*`/swap/`*client_systemname*

C.7. `/vol` Directory

A volume is a collection of related files dedicated to the same function, such as all files required to run a third-party or unbundled application, or all data associated with a particular project. Volumes are attached to the Sun386i file system by the automounter (`automount`(8)).

To create a volume for an application, an administrator must include an entry in the `/etc/auto.vol` Yellow Pages map on the YP master with the format:

> *application_name*   *system:*`/export/`*application_name*

and must then rebuild `auto.vol` by becoming root and using the commands:

```
cd /var/yp
make
```

The automounter (automount(8)) takes references to /vol/*application_name* and uses the entry corresponding to *application_name* in the /etc/auto.vol Yellow Pages map to mount the volume on a temporary mount point under /tmp_mnt. /vol subdirectories do not exist as part of the file system on disk, but rather are created only after automount(8) references to them are made. (*Sun386i Advanced Administration* and the automount(8) man page describe the automounter in more detail.)

If the auto.vol entry indicates that the volume is on a remote system, the automounter creates a temporary mount point under /tmp_mnt and uses this point to mount the remote volume onto the local system via NFS. The automounter returns a symbolic link to the mount point.

If the volume is on the local system, the automounter returns a symbolic link to the volume.

If the application has been exported, is accessible from a volume, and has a link in /files<*n*>/local.unix/bin, any user on the network can execute the application simply by entering *application_name*; users' .login and .cshrc files need no modification, and the application's full path name is not needed. *Sun386i Advanced Administration* provides details.

/vol/local

Whenever a reference to /vol/local is made, for instance, whenever a user logs in (/vol/local/bin and /vol/local/bin.*arch* are included in every user's default $PATH), the automounter mounts /files<*n*>/local.unix and the C-shell builds a table of entries in the user's path. An *application_name* entry in /files<*n*>/local.unix/bin ensures easy network-wide access of your application. *Sun386i Advanced Administration* contains specifics.

C.8. Application Directory Structure

This section describes the preferred subdirectories that your installation script should create under the /usr/local/*application_name* directory. Using this scheme enables your application to work even when users and administrators move it to a different location, and negates the need for users to alter their .login and .cshrc files to use your application. For an explanation of how to release your software so that users can easily load it with snap(1), see pages 144–145.

You should include three major subdirectories for each application:

<arch>.<OS release>

Place architecture and operating system dependent files in *<arch>.<OS release>*, using the subdirectories:

- bin — for binary files
- etc — for configuration information
- lib — for libraries needed at run time, such as shared libraries

where *<arch>* can be either sun1, sun2, sun3, sun4, or sun386 and *<OS release>* has the format SunOS4.0, SunOS4.1BETA1, SunOS4.1BETA2, and so on.

share

Place architecture and operating system independent files in `share`, using the subdirectories:

- `data` — for miscellaneous data files
- `fonts` — for font files
- `icons` — for SunView `.icon` files
- `images` — for SunView `.image` files
- `scripts`— for shell scripts

language

Place a subdirectory in `language` for each language supported:

- `USA-English`
- `English`
- `French`
- `French_Swiss`
- `German`
- `German_Swiss`
- `Italian`
- `Swedish`
- `Spanish`

Four standard subdirectories are available for each language directory:

- `doc` — containing on-line documentation for the application in the particular language
- `help` — containing Spot Help and handbook files for the application in the particular language
- `man` — containing `man` pages for the application in the particular language
- `messages` — containing message files for the application in the particular language

D

Common Object File Format (COFF)

D

Common Object File Format (COFF)

This appendix describes the Common Object File Format (COFF) used on the Sun386i system with the SunOS operating system. COFF is the format of the output file produced by the `as(1)` assembler and the `ld(1)` link editor. This appendix provides a complete description of the COFF format; the Sun386i system does not use all of the features described here.

D.1. COFF Features

Some key features of COFF are:

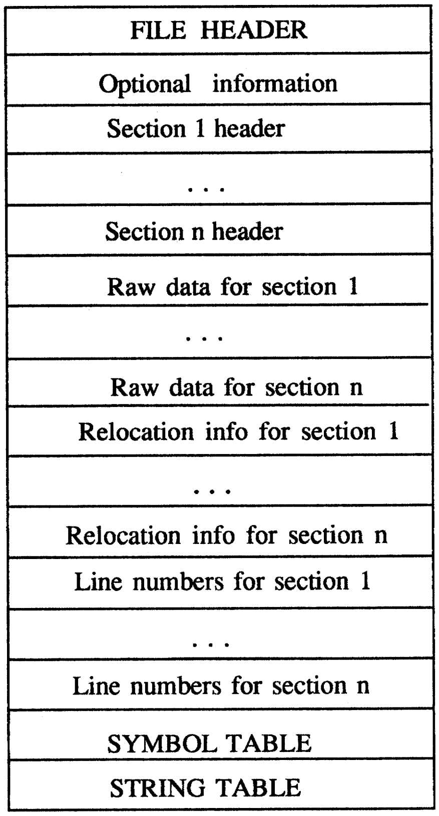
- Applications can add system-dependent information to the object file without causing access utilities to become obsolete.
- Space is provided for symbolic information used by debuggers and other applications.
- Programmers can modify the way the object file is constructed by providing directives at compile time.

The object file supports user-defined sections and contains extensive information for symbolic software testing. An object file contains:

- A file header
- Optional header information
- A table of section headers
- Data corresponding to section headers
- Relocation information
- Line numbers
- A symbol table
- A string table

Figure D-1 on the next page shows the overall structure.

COFF Structure



FILE HEADER

Optional information

Section 1 header

. . .

Section n header

Raw data for section 1

. . .

Raw data for section n

Relocation info for section 1

. . .

Relocation info for section n

Line numbers for section 1

. . .

Line numbers for section n

SYMBOL TABLE

STRING TABLE

Figure D-1 *Object File Format*

The last four sections (relocation, line numbers, symbol table, and string table) may be missing if the program is linked with the `-s` option of the `ld`(1) command, or if the line number information, symbol table, and string table are removed by the `strip`(1) command. The line number information does not appear unless the program is compiled with the `-g` option of the `cc` command. Also, if there are no unresolved external references after linking, the relocation information is no longer needed and is absent.

An object file that contains no errors or unresolved references is considered executable.

D.2. Terms and Conventions

This section explains the COFF-related terms and conventions used throughout this appendix.

Sections
A section is the smallest portion of an object file that is relocated and treated as one separate and distinct entity. In the most common case, there are three sections named `.text`, `.data`, and `.bss`. Additional sections accommodate comments, multiple text or data segments, shared data segments, or user-specified sections. However, the SunOS operating system loads only `.text`, `.data`, and `.bss` into memory when the file is executed.

NOTE *Do not assume that every COFF file will have a certain number of sections. Similarly, do not assume characteristics of sections such as their order, location in the object file, or address at which they are loaded. This information is available only after the object file has been created. Programs manipulating COFF files should obtain this information from file and section headers within the file.*

Physical and Virtual Addresses

The physical address of a section or symbol is the offset of that section or symbol from address zero of the address space. The term "physical address" as used in COFF differs from general use of the term. The physical address of an object is not necessarily the address at which the object is placed when the process is executed. For example, on a system with paging, the address is located with respect to address zero of virtual memory and the system performs another address translation. The section header contains two address fields, a physical address, and a virtual address; but in all versions of COFF on SunOS systems, the physical address is equivalent to the virtual address.

Target Machine

Compilers and link editors produce executable object files that are intended to be run on a particular computer. In the case of cross-compilers, the compilation and link editing are done on one computer with the intent of creating an object file that can be executed on another computer. The term target machine refers to the computer on which the object file is destined to run. In the majority of cases, the target machine and the machine on which the object file is being created are identical.

D.3. File Header

The file header contains the 20 bytes of information shown in Table D-1 below. The last two bytes are flags that are used by `ld` object file utilities.

Table D-1 *File Header Contents*

Bytes	*Declaration*	*Name*	*Description*
0 – 1	`unsigned short`	`f_magic`	Magic number
2 – 3	`unsigned short`	`f_nscns`	Number of sections
4 – 7	`long int`	`f_timdat`	Time and date stamp indicating when the file was created, expressed as the number of elapsed seconds since 00:00:00 GMT, January 1, 1970
8 – 11	`long int`	`f_symptr`	File pointer containing the starting address of the symbol table
12 – 15	`long int`	`f_nsyms`	Number of entries in the symbol table
16 – 17	`unsigned short`	`f_opthdr`	Number of bytes in the optional header
18 – 19	`unsigned short`	`f_flags`	Flags (see Table D-2)

Magic Numbers

The magic number specifies the target machine on which the object file is executable.

Flags

The last two bytes of the file header are flags that describe the type of the object file. Currently defined flags are found in the header file `filehdr.h` and are shown in Table D-2.

Table D-2 *File Header Flags (80286 and 80386 Computers)*

Mnemonic	*Flag*	*Meaning*
`F_RELFLG`	00001	Relocation information stripped from the file
`F_EXEC`	00002	File is executable (that is, no unresolved external references)
`F_LNNO`	00004	Line numbers stripped from the file
`F_LSYMS`	00010	Local symbols stripped from the file
`F_AR16WR`	0000200	16-bit byte-reversed word
`F_AR32WR`	0000400	32-bit byte-reversed word

File Header Declaration

The C structure declaration for the file header, `filehdr.h`, is shown below.

```
struct filehdr
{
     unsigned short  f_magic;   /* magic number */
     unsigned short  f_nscns;   /* number of sec-*/
                                /* tion */
     long            f_timdat;  /* time and date */
                                /* stamp */
     long            f_symptr;  /* file ptr to */
                                /* symbol table */
     long            f_nsyms;   /* # of symbol */
                                /* table entries */
     unsigned short  f_opthdr;  /* optional */
                                /* header size */
     unsigned short  f_flags;   /* flags */
};
#define FILHDR struct filehdr
#define FILHSZ sizeof(FILHDR)
```

D.4. Optional Header Information

The template for optional information varies among different systems that use COFF. Applications place all system-dependent information into this record. This allows different operating systems access to information used only by that particular operating system, without forcing all COFF files to save space for that information. General utility programs (for example, the symbol table access library functions, the disassembler, and so on) are made to work properly on any common object file. This is done by seeking past this record using the size of optional header information in the file header field `f_opthdr`.

Standard SunOS System a.out Header

By default, files produced by the link editor for a SunOS system always have a standard SunOS system `a.out` header in the optional header field. The SunOS system `a.out` header is 28 bytes long. The fields of the optional header are described in Table D-3 (on the next page).

Table D-3 *Optional Header Contents (80286 and 80386 Computers)*

Bytes	*Declaration*	*Name*	*Description*
0 – 1	`short`	`magic`	Magic number
2 – 3	`short`	`vstamp`	Version stamp
4 – 7	`long int`	`tsize`	Size of text in bytes
8 – 11	`long int`	`dsize`	Size of initialized data in bytes
12 – 15	`long int`	`bsize`	Size of uninitialized data in bytes
16 – 19	`long int`	`entry`	Entry point
20 – 23	`long int`	`text_start`	Base address of text
24 – 27	`long int`	`data_start`	Base address of data

Whereas the magic number in the file header specifies the machine on which the object file runs, the magic number in the optional header tells the operating system on that machine how that file should be executed. The magic numbers recognized by the System V/286 and System V/386 operating systems are given in Table D-4.

Table D-4 *System Magic Numbers (80286 and 80386 Computers)*

Value	*Meaning*
0407	Text segment is not write-protected or sharable; data segment is contiguous with the text segment.
0410	Data segment starts at the next segment following the text segment and the text segment is write-protected.
0413	Text and data segments are aligned within `a.out` so it can be directly paged.

Optional Header Declaration

The C language structure declaration currently used for the SunOS system `a.out` file header is shown below. This declaration is in the header file `aouthdr.h`.

```
typedef struct aouthdr
{
        short  magic;       /* magic number */
        short  vstamp;      /* version stamp */
        long   tsize;       /* text size in */
                            /* bytes, padded */
                            /* to full word */
                            /* boundary */
        long   dsize;       /* initialized data */
                            /* size */
        long   bsize;       /* uninitialized  */
                            /* data size */
        long   entry;       /* entry point */
        long   text_start;  /* base of text for */
                            /* this file */
```

```
                    long   data_start; /* base of data for */
                                       /* this file */
        }  AOUTHDR;
```

D.5. Section Headers

Every object file has a table of section headers to specify the layout of data within the file. The section header table consists of one entry for every section in the file. The information in the section header is described in Table D-5.

Table D-5 *Section Header Contents*

Bytes	*Declaration*	*Name*	*Description*
0 – 7	char	s_name	8-character null-padded section name
8 – 11	long int	s_paddr	Physical address of section
12 – 15	long int	s_vaddr	Virtual address of section
16 – 19	long int	s_size	Section size in bytes
20 – 23	long int	s_scnptr	File pointer to raw data
24 – 27	long int	s_relptr	File pointer to relocation entries
28 – 31	long int	s_lnnoptr	File pointer to line number entries
32 – 33	unsigned short	s_nreloc	Number of relocation entries
34 – 35	unsigned short	s_nlnno	Number of line number entries
36 – 39	long int	s_flags	Flags (see Table D-6)

The size of a section is padded to a multiple of four bytes. File pointers are byte offsets that can be used to locate the start of data, relocation, or line number entries for the section. You can use file pointers with the SunOS system function fseek(3S).

Flags

The lower two bytes of the flag field indicate a section type. The flags are described in Table D-6 on the next page.

Table D-6 *Section Header Flags*

Mnemonic	***Flag***	***Meaning***
`STYP_REG`	0x00	Regular section (allocated, relocated, loaded)
`STYP_DSECT`	0x01	Dummy section (not allocated, relocated, not loaded)
`STYP_NOLOAD`	0x02	No load section (allocated, relocated, not loaded)
`STYP_GROUP`	0x04	Grouped section (formed from input sections)
`STYP_PAD`	0x08	Padding section (not allocated, not relocated, loaded)
`STYP_COPY`	0x10	Copy section (for a decision function used in updating fields; not allocated, not relocated, loaded; relocation and line number entries processed normally)
`STYP_TEXT`	0x20	Section contains executable text
`STYP_DATA`	0x40	Section contains initialized data
`STYP_BSS`	0x80	Section contains only uninitialized data
`STYP_INFO`	0x200	Comment section (not allocated, not relocated, not loaded)
`STYP_OVER`	0x400	Overlay section (relocated, not allocated, not loaded)
`STYP_LIB`	0x800	For `.lib` section (treated like `STYP_INFO`)

Section Header Declaration

The C structure declaration for the section headers is described below. This declaration is in the header file `scnhdr.h`.

```
struct scnhdr
{
    char           s_name[8];   /* section name */
    long           s_paddr;     /* phys address */
    long           s_vaddr;     /* virt address */
    long           s_size;      /* section size */
    long           s_scnptr;    /* file ptr to */
                                /* section raw */
                                /* data */
    long           s_relptr     /* file ptr to */
                                /* relocation */
    long           s_lnnoptr;   /* file ptr to */
                                /* line number
    unsigned short s_nreloc;    /* number of */
                                /* relocation */
                                /* entries */
    unsigned short s_nlnno;     /* number of*/
                                /* line number */
                                /* entries */
    long           s_flags;     /* flags */

};
#define SCNHDR struct scnhdr
#define SCNHSZ sizeof(SCNHDR)
```

.bss Section Header

The one deviation from the normal rule in the section header table is the entry for uninitialized data in a .bss section. A .bss section has a size and symbols that refer to it, and symbols that are defined in it. At the same time, a .bss section has no relocation entries, no line number entries, and no data. Therefore, a .bss section has an entry in the section header table but occupies no space elsewhere in the file. In this case, the number of relocation and line number entries, as well as all file pointers in a .bss section header, are zero. The same is true of the STYP_NOLOAD and STYP_DSECT sections.

D.6. Sections

Figure D-1 on page 212 shows that section headers are followed by the appropriate number of bytes of text or data. The raw data for each section begins on a four-byte boundary in the file.

Link editor SECTIONS directives allow users to, among other things:

- Describe how input sections are to be combined
- Direct the placement of output sections
- Rename output sections

If no SECTIONS directives are given, each input section appears in an output section of the same name. For example, if a number of object files, each with a .text section, are linked together, the output object file contains a single .text section made up of the combined input .text sections.

D.7. Relocation Information

Object files have one relocation entry for each relocatable reference in the text or data. The relocation information consists of entries with the format described in Table D-7.

Table D-7 *Relocation Section Contents*

Bytes	*Declaration*	*Name*	*Description*
0 – 3	long int	r_vaddr	(Virtual) address of reference
4 – 7	long int	r_symndx	Symbol table index
8 – 9	unsigned short	r_type	Relocation type

The first four bytes of the entry are the virtual address of the text or data to which this entry applies. The next field is the index, counted from 0, of the symbol table entry that is being referenced. The type field indicates the type of relocation to be applied.

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated. The currently recognized types are given in Table D-8 on the next page.

Table D-8 *Relocation Types (80286 and 80386 Computers)*

Mnemonic	*Flag*	*Meaning*
`R_ABS`	0	Reference is absolute; no relocation is necessary. The entry will be ignored.
`R_DIR16 *`	01	Direct, 16-bit reference to a symbol's virtual address.
`R_REL16 *`	02	"PC-relative," 16-bit reference to a symbol's virtual address. Relative references occur in instructions such as jumps and calls.
`R_DIR32`	06	Direct, 16-bit reference to the symbol's virtual address.
`R_SEG12 *`	011	Direct, 16-bit reference to the segment-selector bits of a 32-bit virtual address.
`R_PCRLONG +`	024	"PC-relative," 32-bit reference to a symbol's virtual address.

* 80286 computers only
+ 80386 computers only

Relocation Entry Declaration

The structure declaration for relocation entries is given below. This declaration is in the header file `reloc.h`.

```
struct reloc
{
  long           r_vaddr; /* virtual address of */
                          /* reference */
  long           r_symndx;/* index into symbol */
                          /* table */
  unsigned short r_type;  /* relocation type */

};
#define RELOC   struct reloc
#define RELSZ   10
```

D.8. Line Numbers

When invoked with either the `-g` or `-go` option, the compiler places an entry in the object file for every source line where a breakpoint can be inserted. You can then reference line numbers when using a software debugger such as `dbx`. All line numbers in a section are grouped by function as shown in Figure D-2.

Symbol index	0
Physical address	Line number
Physical address	Line number
.	.
.	.
.	.
Symbol index	0
Physical address	Line number
Physical address	Line number

Figure D-2 *Line Number Grouping*

The first entry in a function grouping (line number 0) has, in place of the physical address, an index into the symbol table for the entry containing the function name. Subsequent entries have actual line numbers and addresses of the text corresponding to the line numbers. The line number entries are relative to the beginning of the function and appear in increasing order of address.

Line Number Declaration

The structure declaration currently used for line number entries is shown below.

```
struct lineno
{
       union
       {
             long     l_symndx; /* symtbl index */
                                /* of func name */
             long     l_paddr;  /* paddr of line */
                                /* number */
} l_addr;
unassigned short      l_lnno;   /* line number */
};
#define LINENO     struct lineno
#define LINESZ     6
```

D.9. Symbol Table

Because of symbolic debugging requirements, the order of symbols in the symbol table is very important. Symbols appear in the sequence shown in Figure D-3 on the next page.

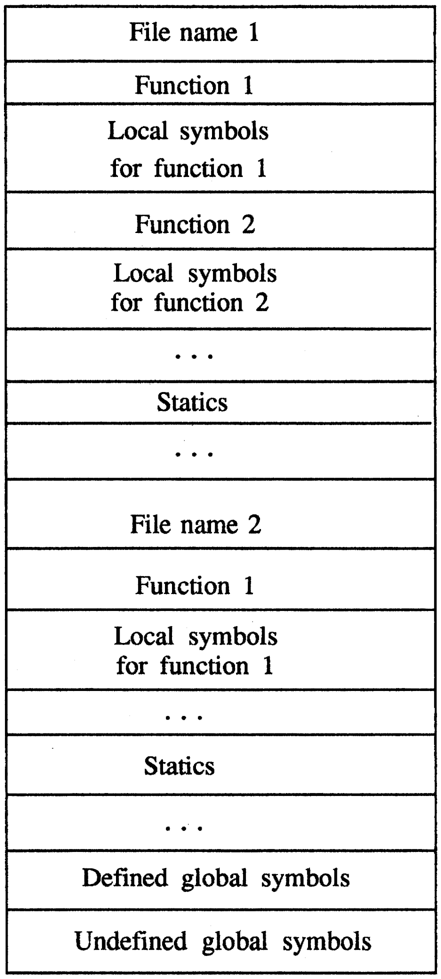


Figure D-3 *COFF Symbol Table*

The word "Statics" in Figure D-3 means symbols defined with the C language storage class `static` outside any function. The symbol table consists of at least one fixed-length entry per symbol with some symbols followed by auxiliary entries of the same size. The entry for each symbol is a structure that holds the value, the type, and other information.

Special Symbols

The symbol table contains some special symbols that are generated by `as`(1) and other tools. (Only the first four symbols, `.file`, `.text`, `.data`, and `.bss` are generated on the Sun386i system.) These symbols are given in Table D-9 on the next page.

Table D-9 *Special Symbols in the Symbol Table*

Symbol	*Meaning*
`.file`	File name
`.text`	Address of `.text` section
`.data`	Address of `.data` section
`.bss`	Address of `.bbs` section
`.bb`	Address of start of inner block
`.eb`	Address of end of inner block
`.bf`	Address of start of function
`.ef`	Address of end of function
`.target`	Pointer to structure or union returned by a function
`.xfake`	Dummy tag name for structure, union, or enumeration
`.eos`	End of members of structure, union, or enumeration
`etext`	Next available address after the end of the output section `.text`
`edata`	Next available address after the end of the output section `.data`
`end`	Next available address after the end of the output section `.bss`

Six of these special symbols occur in pairs. The `.bb` and `.eb` symbols indicate the boundaries of inner blocks; a `.bf` and `.ef` symbol pair brackets each function. A `.xfake` and `.eos` symbol pair names and defines the limit of structures, unions, and enumerations that were named. The `.eos` symbol also appears after named structures, unions, and enumerations.

Symbols and Functions

For each function, the special symbol `.bf` is put between the function name and the first local symbol of the function in the symbol table. Also, the special symbol `.ef` is put immediately after the last local symbol of the function in the symbol table.

The sequence is shown in Figure D-4.

Function name
`.bf`
Local symbol
`.ef`

Figure D-4 *Symbols for Functions*

Symbol Table Entries

All symbols, regardless of storage class and type, have the same format for their entries in the symbol table. The symbol table entries each contain 18 bytes of information. The meaning of each of the fields in the symbol table entry is described in Table D-10. Note that indices for symbol table entries begin at 0 and count upward. Each auxiliary entry also counts as one symbol.

Table D-10 *Symbol Table Entry Format*

Bytes	***Declaration***	***Name***	***Description***
0 – 7	(see text below)	`_n`	These 8 bytes contain either a symbol name or an index to a symbol
8 – 11	`long int`	`n_value`	Symbol value; storage-class dependent
12 – 13	`short`	`n_scnum`	Section number of symbol
14 – 15	`unsigned short`	`n_type`	Basic and derived type specification
16	`char`	`n_sclass`	Storage class of symbol
17	`char`	`n_numaux`	Number of auxiliary entries

Symbol Names

The first eight bytes in the symbol table entry are a union of a character array and two long integers. If the symbol name is eight characters or less, the (null-padded) symbol name is stored there. If the symbol name is longer than eight characters, then the entire symbol name is stored in the string table. In this case, the eight bytes contain two long integers; the first is zero, and the second is the offset (relative to the beginning of the string table) of the name in the string table. Since there can be no symbols with a null name, the zeroes on the first four bytes distinguish a symbol table entry with an offset from one with a name in the first eight bytes as shown in Table D-11.

If the program is compiled with the `-g` option, the symbol is stored like a `long` (greater than eight characters) symbol, but only the first byte is 0. The remaining three bytes of the first `long` word are used to store `dbx` information.

Table D-11 *Name Field*

Bytes	***Declaration***	***Name***	***Description***
0 – 7	`char`	`n_name`	8-character null-padded symbol name
0 – 3	`long`	`n_zeroes`	Zero in this field indicates the name is in the string table
0	`char`	`n_leading_zero`	Null character
1	`char`	`n_dbx_type`	Symbol table type
2 – 3	`short`	`n_dbx_desc`	Value of description field
4 – 7	`long`	`n_offset`	Offset of the name in the string table

Special symbols generated by the C compiler are discussed in the Special Symbols section starting on page 221.

Storage Classes

The storage class field has one of the values described in Table D-12 on the following page. These `#defines` are in the header file `storclass.h`.

Table D-12 *Storage Classes*

Mnemonic	*Value*	*Storage Class*
`C_EFCN`	–1	Physical end of a function
`C_NULL`	0	—
`C_AUTO`	1	Automatic variable
`C_EXT`	2	External symbol
`C_STAT`	3	Static
`C_REG`	4	Register variable
`C_EXTDEF`	5	External definition
`C_LABEL`	6	Label
`C_ULABEL`	7	Undefined label
`C_MOS`	8	Member of structure
`C_ARG`	9	Function argument
`C_STRTAG`	10	Structure tag
`C_MOU`	11	Member of union
`C_UNTAG`	12	Union tag
`C_TPDEF`	13	Type definition
`C_USTATIC`	14	Uninitialized static
`C_ENTAG`	15	Enumeration tag
`C_MOE`	16	Member of enumeration
`C_REGPARM`	17	Register parameter
`C_FIELD`	18	Bit field
`C_BLOCK`	100	Beginning and end of block
`C_FCN`	101	Beginning and end of function
`C_EOS`	102	End of structure
`C_FILE`	103	File name
`C_LINE`	104	Used only by utility programs
`C_ALIAS`	105	Duplicated tag
`C_HIDDEN`	106	Like static, used to avoid name conflicts

All of these storage classes except for `C_ALIAS` and `C_HIDDEN` are generated by the `cc`(1) or `as`(1) commands. The storage class `C_HIDDEN` is not used by any SunOS system tools.

Some of these storage classes are used only internally by the C compiler. These storage classes are `C_EFCN`, `C_EXTDEF`, `C_ULABEL`, `C_USTATIC`, and `C_LINE`.

Storage Classes for Special Symbols

Some special symbols are restricted to certain storage classes. They are given in Table D-13 on the next page.

Table D-13 *Storage Class by Special Symbol*

Special Symbol	*Storage Class*
.file	C_FILE
.bb	C_BLOCK
.eb	C_BLOCK
.bf	C_FCN
.ef	C_FCN
.target	C_AUTO
.*x*fake	C_STRTAG, C_UNTAG, C_ENTAG
.eos	C_EOS
.text	C_STAT
.data	C_STAT
.bss	C_STAT

Also, some storage classes are used only for certain special symbols. They are summarized in Table D-14.

Table D-14 *Restricted Storage Classes*

Storage Class	*Special Symbol*
C_BLOCK	.bb, .eb
C_FCN	.bf, .ef
C_EOS	.eos
C_FILE	.file

Symbol Value Field

The meaning of the value of a symbol depends on its storage class. This relationship is summarized in Table D-15 on the following page.

Table D-15 *Storage Class and Value*

Storage Class	*Meaning of Value*
C_AUTO	Stack offset in bytes
C_EXT	Relocatable address
C_STAT	Relocatable address
C_REG	Register number
C_LABEL	Relocatable address
C_MOS	Offset in bytes
C_ARG	Stack offset in bytes
C_STRTAG	0
C_MOU	0
C_UNTAG	0
C_TPDEF	0
C_ENTAG	0
C_MOE	Enumeration value
C_REGPARM	Register number
C_FIELD	Bit displacement
C_BLOCK	Relocatable address
C_FCN	Relocatable address
C_EOS	Size
C_FILE	(See text below)
C_ALIAS	Tag index
C_HIDDEN	Relocatable address

If a symbol has storage class C_FILE, the value of that symbol equals the symbol table entry index of the next .file symbol. That is, the .file entries form a one-way linked list in the symbol table. If there are no more .file entries in the symbol table, the value of the symbol is the index of the first global symbol.

Relocatable symbols have a value equal to the virtual address of that symbol. When the section is relocated by the link editor, the value of these symbols changes.

Section Number Field

Section numbers are listed in Table D-16 below.

Table D-16 *Section Number*

Mnemonic	*Section Number*	*Meaning*
N_DEBUG	–2	Special symbolic debugging symbol
N_ABS	–1	Absolute symbol
N_UNDEF	0	Undefined external symbol
N_SCNUM	1 – 077777	Section number where symbol is defined

A special section number (–2) marks symbolic debugging symbols, including structure/union/enumeration tag names, typedefs, and the name of the file. A section number of –1 indicates that the symbol has a value but is not relocatable. Examples of absolute-valued symbols include automatic and register variables, function arguments, and .eos symbols.

With one exception, a section number of 0 indicates a relocatable external symbol that is not defined in the current file. The one exception is a multiply defined external symbol (such as FORTRAN common or an uninitialized variable defined external to a function in C). In the symbol table of each file where the symbol is defined, the section number of the symbol is 0, and the value of the symbol is a positive number giving the size of the symbol. When the files are combined to form an executable object file, the link editor combines all the input symbols of the same name into one symbol with the section number of the .bss section. The maximum size of all the input symbols with the same name is used to allocate space for the symbol and the value becomes the address of the symbol. This is the only case where a symbol has a section number of 0 and a nonzero value.

Section Numbers and Storage Classes

Symbols having certain storage classes are also restricted to certain section numbers. They are summarized in Table D-17.

Table D-17 *Section Number and Storage Class*

Storage Class	*Section Number*
C_AUTO	N_ABS
C_EXT	N_ABS, N_UNDEF, N_SCNUM
C_STAT	N_SCNUM
C_REG	N_ABS
C_LABEL	N_UNDEF, N_SCNUM
C_MOS	N_ABS
C_ARG	N_ABS
C_STRTAG	N_DEBUG
C_MOU	N_ABS
C_UNTAG	N_DEBUG
C_TPDEF	N_DEBUG
C_ENTAG	N_DEBUG
C_MOE	N_ABS
C_REGPARM	N_ABS
C_FIELD	N_ABS
C_BLOCK	N_SCNUM
C_FCN	N_SCNUM
C_EOS	N_ABS
C_FILE	N_DEBUG
C_ALIAS	N_DEBUG

Type Entry

The type field in the symbol table entry contains information about the basic and derived type for the symbol. This information is generated by the C compiler only if the -g or -go option is used. Each symbol has exactly one basic or fundamental type but can have more than one derived type. The format of the 16-bit type entry is shown below.

d6	d5	d4	d3	d2	d1	typ

Figure D-5 *16-Bit Type Entry Format*

Bits 0 through 3, called `typ`, indicate one of the fundamental types given in Table D-18.

Table D-18 *Fundamental Types*

Mnemonic	***Value***	***Type***
`T_NULL`	0	Type not assigned
`T_ARG`	1	Function argument (used only by compiler)
`T_CHAR`	2	Character
`T_SHORT`	3	Short integer
`T_INT`	4	Integer
`T_LONG`	5	Long integer
`T_FLOAT`	6	Floating point
`T_DOUBLE`	7	Double word
`T_STRUCT`	8	Structure
`T_UNION`	9	Union
`T_ENUM`	10	Enumeration
`T_MOE`	11	Member of enumeration
`T_UCHAR`	12	Unsigned character
`T_USHORT`	13	Unsigned short
`T_UINT`	14	Unsigned integer
`T_ULONG`	15	Unsigned long

Bits 4 through 15 are arranged as six 2-bit fields marked `d1` through `d6`. These `d` fields represent levels of the derived types given in Table D-19.

Table D-19 *Derived Types*

Mnemonic	***Value***	***Type***
`DT_NON`	0	No derived type
`DT_PTR`	1	Pointer
`DT_FCN`	2	Function
`DT_ARY`	3	Array

The following examples demonstrate the interpretation of the symbol table entry representing type.

```
char *func();
```

Here func is the name of a function that returns a pointer to a character. The fundamental type of func is 2 (character), the d1 field is 2 (function), and the d2 field is 1 (pointer). Therefore, the type word in the symbol table for func contains the hexadecimal number 0x62, which is interpreted to mean a function that returns a pointer to a character.

```
short *tabptr[10] [25] [3];
```

Here tabptr is a three-dimensional array of pointers to short integers. The fundamental type of tabptr is 3 (short integer); the d1, d2, and d3 fields each contains a 3 (array), and the d4 field is 1 (pointer). Therefore, the type entry in the symbol table contains the hexadecimal number 0x7f3 indicating a three-dimensional array of pointers to short integers.

Type Entries and Storage Classes

Table D-20 shows the type entries that are legal for each storage class.

Table D-20 *Type Entries by Storage Class*

Storage	*d Entry*			*typ Entry*
Class	*Function?*	*Array?*	*Pointer?*	*Basic Type*
C_AUTO	no	yes	yes	Any except T_MOE
C_EXT	yes	yes	yes	Any except T_MOE
C_STAT	yes	yes	yes	Any except T_MOE
C_REG	no	no	yes	Any except T_MOE
C_LABEL	no	no	no	T_NULL
C_MOS	no	yes	yes	Any except T_MOE
C_ARG	yes	no	yes	Any except T_MOE
C_STRTAG	no	no	no	T_STRUCT
C_MOU	no	yes	yes	Any except T_MOE
C_UNTAG	no	no	no	T_UNION
C_TPDEF	no	yes	yes	Any except T_MOE
C_ENTAG	no	no	no	T_ENUM
C_MOE	no	no	no	T_MOE
C_REGPARM	no	no	yes	Any except T_MOE
C_FIELD	no	no	no	T_ENUM, T_UCHAR, T_USHORT, T_UNIT, T_ULONG
C_BLOCK	no	no	no	T_NULL
C_FCN	no	no	no	T_NULL
C_EOS	no	no	no	T_NULL
C_FILE	no	no	no	T_NULL
C_ALIAS	no	no	no	T_STRUCT, T_UNION, T_ENUM

Conditions for the d entries apply to d1 through d6, except that it is impossible to have two consecutive derived types of function.

Although function arguments can be declared as arrays, they are changed to pointers by default. Therefore, no function argument can have an array as its first derived type.

Structure for Symbol Table Entries

The C language structure declaration for the symbol table entry is given below. This declaration is in the header file `syms.h`.

```
struct syment
{
  union
  {
         char     _n_name[SYMNMLEN]; /* symbol */
                                     /* name */
         struct
         {
            long  _n_zeroes;    /* symbol name */
            long  _n_offset;    /* location in */
                                /* string table */
         }_n_n;
         char     *_n_nptr[2];   /* allows */
                                /* overlaying */
         struct
         {
            char  _n_leading_zero; /* null */
                                   /*char */
            char  _n_dbx_type;     /* symbol */
                                   /* tbl type */
            short _n_dbx_desc;    /* value of */
                                  /* desc field */
                                  /* in string */
            long  _n_stab_ptr;    /* table ptr */
          } _n_dbx;
  } _n;
  long              n_value;          /* symbol */
                                      /* value */
  short             n_scnum;          /* section */
                                      /* number */
  unsigned short    n_type;           /* type and */
                                      /* derived */
  char              n_sclass;         /* storage */
                                      /* class */
  char              n_numaux;         /* number */
                                      /* of aux */
                                      /* entries */
};

#define n_name     _n._n_name
#define n_nptr     _n._n_nptr[1]
```

```
#define n_zeroes   _n._n_n._n_zeroes
#define n_offset   _n._n_n._n_offset
#define n_leading_zero _n._n_dbx._n_leading_zero
#define n_dbx_type _n._n_dbx._n_dbx_type
#define n_dbx_desc _n._n_dbx._n_dbx_desc
#define SYMNMLEN  8
#define SYMESZ   18  /* size of a symbol table */
                     /* entry */
```

Auxiliary Table Entries

An auxiliary table entry of a symbol contains the same number of bytes as the symbol table entry. However, unlike symbol table entries, the format of an auxiliary table entry of a symbol depends on its type and storage class. They are summarized in Table D-21.

Table D-21 *Auxiliary Symbol Table Entries*

	Storage	*Type Entry*		*Auxiliary*
Name	*Class*	*d*	*typ*	*Entry Format*
.file	C_FILE	DT_NON	T_NULL	File name
.text, .data, .bss	C_STAT	DT_NON	T_NULL	Section
tagname	C_STRTAG C_UNTAG C_ENTAG	DT_NON	T_NULL	Tag name
.eos	C_EOS	DT_NON	T_NULL	End of structure
fcname	C_EXT C_STAT	DT_FCN	(Note 1)	Function
arrname	(Note 2)	DT_ARY	(Note 1)	Array
.bb, .eb	C_BLOCK	DT_NON	T_NULL	Beginning and end of block
.bf, .ef	C_FCN	DT_NON	T_NULL	Beginning and end of function
name related to structure, union, enumeration	(Note 2)	DT_PTR DT_ARR DT_NON	T_STRUCT, T_UNION, T_ENUM	Name related to structure, union, enumeration

NOTES

1. Any except T_MOE.

2. C_AUTO, C_STAT, C_MOS, C_MOU, C_TPDEF.

In Table D-21, *tagname* means any symbol name including the special symbol *.x*fake, and *fcname* and *arrname* represent any symbol name for a function or an array respectively. Any symbol that satisfies more than one condition in Table D-21 should have a union format in its auxiliary entry.

Do not assume the number of auxiliary entries associated with any given symbol table entry. Instead, check the n_numaux *field in the symbol table for this information.*

File Names

Each of the auxiliary table entries for a file name contains a 14-character file name in bytes 0 through 13. The remaining bytes are 0.

Sections

The auxiliary table entries for sections have the format shown in Table D-22.

Table D-22 *Format for Auxiliary Table Entries for Sections*

Bytes	***Declaration***	***Name***	***Description***
0 – 3	`long int`	`x_scnlen`	Section length
4 – 5	`unsigned short`	`x_nreloc`	Number of relocation entries
6 – 7	`unsigned short`	`x_nlinno`	Number of line numbers
8 – 17	—	—	Unused (filled with zeroes)

Tag Names

The auxiliary table entries for tag names have the format shown in Table D-23.

Table D-23 *Tag Name Table Entries*

Bytes	***Declaration***	***Name***	***Description***
0 – 5	—	—	Unused (filled with zeroes)
6 – 7	`unsigned short`	`x_size`	Size of structure, union, and enumeration
8 – 11	—	—	Unused (filled with zeroes)
12 – 15	`long int`	`x_endndx`	Index of next entry beyond this structure, union, or enumeration
16 – 17	—	—	Unused (filled with zeroes)

End of Structures

The auxiliary table entries for the end of structures have the format shown in Table D-24.

Table D-24 *Table Entries for End of Structures*

Bytes	***Declaration***	***Name***	***Description***
0 – 3	`long int`	`x_tagndx`	Tag index
4 – 5	—	—	Unused (filled with zeroes)
6 – 7	`unsigned short`	`x_size`	Size of structure, union, or enumeration
8 – 17	—	—	Unused (filled with zeroes)

Functions

The auxiliary table entries for functions have the format shown in Table D-25 on the next page.

Table D-25 *Table Entries for Functions*

Bytes	*Declaration*	*Name*	*Description*
0 – 3	long int	x_tagndx	Tag index
4 – 7	long int	x_fsize	Size of function (in bytes)
8 – 11	long int	x_lnnoptr	File pointer to line number
12 – 15	long int	x_endndx	Index of next entry beyond this point
16 – 17	unsigned short	x_tvndx	Index of function's address in the transfer vector table (not used in the SunOS system)

Arrays

The auxiliary table entries for arrays have the format shown in Table D-26. Defining arrays having more than four dimensions produces a warning message.

Table D-26 *Table Entries for Arrays*

Bytes	*Declaration*	*Name*	*Description*
0 – 3	long int	x_tagndx	Tag index
4 – 5	unsigned short	x_lnno	Line number of declaration
6 – 7	unsigned short	x_size	Size of array
8 – 9	unsigned short	x_dimen [0]	First dimension
10 – 11	unsigned short	x_dimen [1]	Second dimension
12 – 13	unsigned short	x_dimen [2]	Third dimension
14 – 15	unsigned short	x_dimen [3]	Fourth dimension
16 – 17	—	—	Unused (filled with zeroes)

Beginning of Blocks and Functions

The auxiliary table entries for the beginning of blocks and functions have the format shown in Table D-27.

Table D-27 *Format for Beginning of Block and Function Entries*

Bytes	*Declaration*	*Name*	*Description*
0 – 3	—	—	Unused (filled with zeroes)
4 – 5	unsigned short	x_lnno	C-source line number
6 – 11	—	—	Unused (filled with zeroes)
12 – 15	long int	x_endndx	Index of next entry past this block
16 – 17	—	—	Unused (filled with zeroes)

End of Blocks and Functions

The auxiliary table entries for the end of blocks and functions have the format shown in Table D-28 on the following page.

Table D-28 *End of Block and Function Entries*

Bytes	*Declaration*	*Name*	*Description*
0 – 3	—	—	Unused (filled with zeroes)
4 – 5	`unsigned short`	`x_lnno`	C-source line number
6 – 17	—	—	Unused (filled with zeroes)

Names Related to Structures, Unions, and Enumerations

The auxiliary table entries for structure, union, and enumeration symbols have the format shown in Table D-29.

Table D-29 *Entries for Structures, Unions, and Enumerations*

Bytes	*Declaration*	*Name*	*Description*
0 – 3	`long int`	`x_tagndx`	Tag index
4 – 5	—	—	Unused (filled with zeroes)
6 – 7	`unsigned short`	`x_size`	Size of the structure, union, or enumeration
8 – 17	—		Unused (filled with zeroes)

Aggregates defined by `typedef` may or may not have auxiliary table entries, as shown in the example below.

```
typedef struct people STUDENT;
struct people
{
        char name[20];
        long id;
};
typedef struct people EMPLOYEE;
```

The symbol `EMPLOYEE` has an auxiliary table entry in the symbol table but symbol `STUDENT` does not because it is a forward reference to a structure.

Auxiliary Entry Declaration

The C language structure declaration for an auxiliary symbol table entry is shown below. This declaration is in the header file `syms.h`.

```
union auxent
{
       struct
       {
              long x_tagndx;
              union
              {
                  struct
                  {
                          unsigned short x_lnno;
                          unsigned short x_size;
```

```
                } x_lnsz;
                long x_fsize;
            } x_misc;
            union
            {
                struct
                {
                        long x_lnnoptr;
                        long x_endndx;
                } x_fcn;
                struct
                {
                        unsigned short
                        x_dimen[DIMNUM];
                } x_ary;
            }x_fcnary;
            unsigned short x_tvndx;
      } x_sym;
      struct
      {
            char x_fname[FILNMLEN];
      } x_file;
      struct
      {
            long x_scnlen;
            unsigned short x_nreloc;
            unsigned short x_nlinno;
      }x_scn;
      struct
      {
            long x_tvfill;
            unsigned short x_tvlen;
            unsigned short x_tvran[2];
      } x_tv;
}
#define FILNMLEN 14
#define DIMNUM   4
#define AUXENT union auxent
#define AUXESZ   18
```

D.10. String Table

Symbol table names longer than eight characters are stored contiguously in the string table with each symbol name delimited by a null byte. The first four bytes of the string table are the size of the string table in bytes; offsets into the string table, therefore, are greater than or equal to 4. For example, given a file containing two symbols (with names longer than eight characters, `long_name_1` and `another_one`) the string table has the format as shown in Table D-30.

Table D-30 *String Table*

'l'	'o'	'n'	'g'
'_'	'n'	'a'	'm'
'e'	'_'	'1'	'\0'
'a'	'n'	'o'	't'
'h'	'e'	'r'	'_'
'o'	'n'	'e'	'\0'

The index of `long_name_1` in the string table is 4 and the index of `another_one` is 16.

D.11. Access Routines

SunOS system releases contain a set of access routines that are used for reading the various parts of a common object file. Although the calling program must know the detailed structure of the parts of the object file it processes, the routines effectively insulate the calling program from the knowledge of the overall structure of the object file.

The access routines can be divided into functions that

- Open or close an object file
- Read header or symbol table information
- Position an object file at the start of a particular section of the object file
- Return the sumbol table index for a particular symbol

These routines are located in the `libld.a` library. These routines are listed and briefly described in Table 4-3 on page 32.

E

Differences Between Sun C and Kernighan and Ritchie C

E

Differences Between Sun C and Kernighan and Ritchie C

This appendix notes the differences between Sun C and the language described in *The C Programming Language*[1] by Brian W. Kernighan and Dennis M. Ritchie (hereafter referred to as K&R). The discussion is organized according to the arrangement of topics (sections and subsections) appearing in Appendix A: C Reference Manual of the referenced work. The numbers in parentheses are the section numbers in that appendix.

E.1. Lexical Conventions (2)

Keywords (2.3)

Sun C has additional keywords: `void` and `enum`.

E.2. What's in a Name? (4)

K&R C provides two name spaces: one for `struct/union` tags and `struct/union` members; the other for all variables, functions, `typedef` names, and so on. Sun C provides separate name spaces for:

- `struct/union` and `enum` tags
- Elements of each different type of `struct/union`
- Everything else: regular variables and functions

E.3. Conversions (6)

Characters and Integers (6.1)

Sun characters are signed, and all 7-bit ASCII characters are positive. Unsigned characters are, of course, unsigned, and promote to `unsigned` (see Type Specifiers on page 240).

`float` and `double` (6.2)

K&R states "...whenever a `float` appears in an expression it is lengthened to `double` by zero-padding its fraction." This is not how floating-point representation is done on the Sun. `float`s are lengthened to `double`s in expressions, but considerable work must be expended to do it, since the exponent part is of a differ-

[1] Prentice-Hall Inc., Englewood Cliffs, New Jersey

ent width and bias.

Sun also provides a compiler option, `-fsingle`, to prevent this widening from happening for any expression involving only `floats`. This flag will not prevent `float` formal parameters from being rewritten as `doubles`, nor `float`-valued actual parameters from being promoted to `doubles`.

Arithmetic Conversions (6.6)

`unsigned char` and `unsigned short` promote to `unsigned`. Since `long==int` on a Sun, nothing ever promotes to `long`.

E.4. Expressions (7)

Primary Expressions (7.1)

K&R does not discuss the possibility of passing `structs` or `unions` as value parameters. This is because it is not allowed in K&R C (see External Function Definitions on page 242). Many people believe that `struct` names are treated as array names, and implicitly converted to "point to `struct`," that is, reference parameters. This may be an extension provided by some implementations, but does not appear to be officially sanctioned. In any case, Sun C supports passing `structs` and `unions` by value.

Multiplicative Operators (7.3)

As on the PDP-11, the sign of the remainder is the same as the sign of the dividend. K&R indicates that `%` may not be applied to operands of type `float` or `double`.

E.5. Declarations (8)

Storage Class Specifiers (8.1)

On the PDP-11, you can assign only `int`, `char`, and pointer types to registers with the `register` storage class specifier. On a Sun system, you can assign any integral type (combinations of `char`, `short`, `int`, `long`, `unsigned`, `enum`) and any pointer type to registers.

Type Specifiers (8.2)

K&R list the scalar types as `char`, `short int`, `int`, `long int`, `float`, and `double`. Sun also supports `unsigned char`, `unsigned short int`, and `enum` types. The additional unsigned types have the attributes one would expect, and promote to `unsigned int` rather than `int` during "the usual arithmetic conversions."

The `enum` type is a sort of enumeration. To declare such a type, write something like `typedef enum { red, green, blue } color`. You can also give explicit values to the enumeration members:

```
typedef enum { walnut =3, almond, brazil, tangerine=0 } nut;
```

The rules are:

1. If no value is specified, the first member has value 0.
2. If no value is specified, each member takes a value one greater than the one preceding it.
3. There is no check for duplicate values.

`Enums` can take a tag, as `structs` do:

```
enum nut { .... };
```

The tag is in the same name space as `struct` and `union` tags. The member names are in the same name space as plain variable and function names. Thus `blue` can only be a member of one enumeration, *or* be a variable or function name.

The current semantics of `enums` are agreed to be ill-conceived. They may be assigned, compared only for equality, passed as parameters, and used as the type of a `switch` expression or `case` label. You cannot use them in arithmetic or as subscripts.

In Sun C, you can declare functions to return the type `void`. This means the function does not return any value, so it is functionally a subroutine. There are no objects of type `void`.

As of the Sun 4.0 software release, pointers may be declared as pointers-to-nothing (`void *`). You cannot dereference these pointers, but you can assign them to other pointers without complaints from the compiler. This is the `opaque` type provided by ANSI C.

Meaning of Declarators (8.4)

K&R prohibits declaring a function returning a `struct` or `union`. Sun C permits it.

Structure and Union Declarations (8.5)

In Sun C, fields are packed left-to-right within a storage unit appropriate to the type they are declared to be. You can declare them as any of the integer types, and `enum`. No matter what their declaration, all fields are unsigned, and thus zero-extended for the purposes of "the normal conversions."

As mentioned in Section E.2 on page 239, all structure members are in the same name space in K&R C. However, elements in different `structs` can have the same name, so long as they also have the same offset and type. This means that you can interpret the element selection operators `.` and `->` without considering the type of the expression on the left, which need not even be a pointer type. (See Structures and Unions on page 243.)

In Sun C, interpretation of `.` and `->` take into account the type of the `struct/union` or pointer expression on the left to determine the name on the right. There can now be apparent clashes between offsets and types between members of different aggregates but having the same name. The only difficulty comes if the type of the left-hand expression does not properly disambiguate the name, in which case:

1. If there is no ambiguity, then the only choice is taken, and a warning is issued.
2. If there is ambiguity, the program is considered to be in error.

E.6. Statements (9)

Switch Statement (9.7)

K&R C implies that the type of a `switch` expression must be an integer type. Sun C accepts `floats` and `doubles` (which are fixed to `ints`) and `enum` types as well.

E.7. External Definitions (10)

External Function Definitions (10.1)

K&R C prohibits passing an argument of type `struct` or `union` to a function, so it is useless to declare a function with a formal parameter of one of those types. Sun C permits `struct/union` value parameters.

E.8. Scope Rules (11)

Lexical Scope (11.1)

K&R indicates that when a variable is redeclared inside a compound statement, the outer declaration is "pushed down" for the duration of the block. That is, it is overridden while in the block, but then resumes force following the block. This is not true for Sun C if the inner declaration is of class `extern`. In this case, the declaration persists until the end of the file; if it redeclares a name with a definition in an outer block, the compiler will complain about redeclaring a variable.

Scope of Externals (11.2)

The linking rules Sun C uses are a bit more liberal than the rules implied by K&R, but are the same as some PDP-11 implementations. Here is description of the Sun linkage rules:

1. C uninitialized global data is treated like FORTRAN uninitialized `COMMON`. To borrow terminology from ANSI C, this constitutes a "tentative definition." C initialized global data is treated like FORTRAN `COMMON` initialized by `BLOCK DATA`. This constitutes a "true definition."
2. A tentative definition in a library module will not cause the module to be loaded. A true definition will cause loading, if the name occurs as a reference or tentative definition in a module that is already being linked. The "already" here is important since order matters.
3. If the linker sees any true definitions of a name among the modules to be linked, such definitions override all tentative definitions. This includes the case where the true definition allocates less space for the named object than the tentative definition(s) would. This is a rather severe shortcoming of the current scheme, and has been around since 1978.
4. If no true definitions of a name are seen, the name is defined by the linker, and space is allocated. The amount of space allocated should be the maximum of the size specified in any of the tentative definitions in the modules being linked.

It is a bug that tentative definitions in library modules are not linked in this size calculation. Again, this has existed since 1978. Note that for this bug to manifest itself, the name in question must be tentatively defined or at least referenced in modules that are being linked.

E.9. Types Revisited

Structures and Unions (14.1)

K&R says you cannot assign `structs/unions` or pass them as parameters, though these operations may be allowed in the future. Sun C allows them now.

Explicit Pointer Conversions (14.4)

On Sun machines, a pointer corresponds to a 32-bit integer. Addresses are measured in 8-bit bytes. Alignment varies depending on the model: on Sun-2 systems, all data bigger than a `char` must be 0mod2 aligned. On Sun-3 systems, data need not be aligned, but performance is improved if `shorts` are aligned 0mod2, and `ints`, `floats`, and `doubles` are aligned 0mod4. On Sun386i systems, data must be aligned on natural boundaries: bytes on byte boundaries, words (16 bits) on word boundaries, and doublewords (32 bits) on doubleword boundaries. Structures are aligned on doublewords on Sun386i systems.

E.10. Constant Expressions (15)

K&R prohibits cast operators as part of constant expressions. Sun C allows them, except in preprocessor constant expressions (see Section 12.3, Conditional Compilation, in K&R Appendix A), where the `sizeof` operator is also prohibited. K&R seemingly permits `sizeof` in preprocessor constant expressions, but the common implementation does not.

E.11. Anachronisms (17)

Sun C will not recognize the anachronisms listed in this section. Use `op=` instead of `=op` for assignment operators and include an equals sign when introducing an initializer.

F

C on the Sun386i System

F

C on the Sun386i System

This appendix describes the operational characteristics of Sun386i C on the Intel 80386 microprocessor. These characteristics are not part of the C language itself, but rather are the results of machine-dependent implementation choices. Code that depends on these features may not be portable; however, knowledge of these features could help when porting C programs from other systems or interfacing C with assembly language programs. This appendix assumes familiarity with the C programming language.

F.1. Names

Variables that are of storage class `extern` appear in the generated assembler output as they were in the C source. Case distinction in such names is preserved. The names of variables defined to be of storage class `static`, `auto`, or `register` do not appear in the generated code. There is effectively no limit to the length of a variable's name.

F.2. Data Types and Sizes

The various basic data types have the following lengths:

`char`	8 bits
`short`	16 bits
`int`	32 bits
`long`	32 bits
`float`	32 bits
pointers	32 bits
`double`	64 bits

There are two kinds of alignment used depending upon the data type and storage class of a variable. Four-byte alignment means that the variable's address will be a multiple of four, two-byte alignment indicates an address that is even.

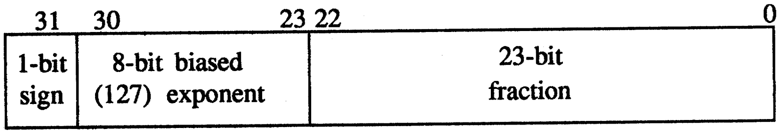
Unless otherwise specified, all variables have four-byte alignment. Excepting those with `extern` or `static` storage class, `short` and `unsigned short` types have two-byte alignment and `char` types are unaligned. Within a structure, individual members of type `char` are unaligned and those of `short` and `unsigned short` are two-byte aligned. A structure always has the alignment properties of its most strongly aligned member. This can result in one, two, or three bytes of padding at the end of a `struct`. Arguments, which are always cast to `int` or `double` if they are arithmetic, are therefore always four-byte aligned.

You can declare bit fields with any integral type, but they are always treated as though they had been declared with an equivalent unsigned type. The bits are allocated from right to left, that is, the low-order bits are allocated first. Data is aligned beginning at the least-significant bit of the word.

F.3. Data Layout

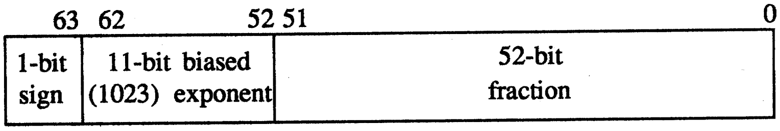
Within a `short`, the low-order byte is at the lower address. Within an `int` (or a `long`), the low-order word is at the lower address.

The floating-point format follows the IEEE standard. Values of type `float` are stored as two-word quantities, interpreted as:



$$\text{value} = (\text{sign})\ (\text{fraction} \times 2^{(e - 127)})$$

Values of type `double` are stored as four-word quantities, interpreted as:



$$\text{value} = (\text{sign})\ (\text{fraction} \times 2^{(e - 1023)})$$

Extended values never occur outside of the numeric processor.

The words are addressed such that the least-significant part of the fraction has the lowest address, and the word containing the sign and exponent has the highest address. The value of the `float` can range from about 8.43^{-37} to 3.37^{38}, that of a `double` from about 4.19^{-307} to 1.67^{308}.

F.4. Initialization

You can initialize scalar variables. You can initialize structures and arrays only if they are storage class `static` or `extern`. You cannot initialize unions.

Three registers (`EDI`, `ESI`, and `EBX`) are available for user variables. The compiler allocates only the types `long`, `int`, `short`, and `char`, their unsigned counterparts, and pointers to registers (and then only on programmer request). Only one `char` register variable is possible.

F.5. Bit Shifting

Right shifts are arithmetic if the value being shifted is signed; they are logical if the value is unsigned. That is, the vacated bits are filled with copies of the sign bit if the value is signed and zeroes if it is unsigned.

When porting code to the Sun386i system, you could run into problems because of a difference in how the 80386 handles bit shifting. The maximum shift count for the Sun386i system is 31, unlike most machines which allow a higher shift count (even though a count above 32 is meaningless). The limit of 31 reduces maximum execution time.

If the instruction `i = y << j` is executed on a Sun386i system and the value of `j` is greater than 31, only the *lower five bits* of `j` are used for the count. This means that a shift of 32 is equivalent to a shift of zero bits. To avoid this problem, you must search through your code and check all shift operations. For each operation such as *value* `<< =` *count*`;` where *count* could be greater than 31, insert the following code:

```
if (count > 31)
   value = 0;
else
   value <<= count;
```

F.6. Structure Return

Functions can return structures, but you should use caution. If you allow a function returning a `struct` to default to `int` by mistake, the argument will be misinterpreted and data could be overwritten. The compiler cannot detect this error, but `lint`(1V) can; this is a good reason to use `lint`.

F.7. Register Usage

`EAX`, `ECX`, and `EDX` are temporary registers. These registers are not saved across subroutine calls.

The compiler never generates code that changes the contents of a segment register. These are assumed to be set correctly when a program begins execution.

`ESP` is used as a stack pointer in the conventional way. `EBP` is used as a frame pointer.

Functions returning integrald-typed values do so in `EAX`. Floating-point return values are in `ST(0)`. Structures are returned in a caller-allocated buffer.

F.8. Stack Format

A stack frame for a function invocation is set up and taken down as follows:

1. The caller pushes the arguments in right-to-left order (last argument first). If the called function returns a `struct`, the caller pushes the address of an appropriate buffer.
2. The return address is pushed onto the stack by executing the call instruction.
3. `EBP` is pushed, and `EBP` is made to point to the stack top, which now contains its old value.
4. The called function then allocates all of its local and temporary space.
5. The called function may push the current values of `EDI`, `ESI`, and `EBX`, depending on whether any of these registers are used by the called function, thereby saving their values for the calling function.
6. When the called function is finished, it restores `EBX`, `ESI`, and `EDI` if they were saved.
7. It then removes its local and temporary space from the stack, reloads the previous value of `EBP`, and returns.

8. The caller pops the arguments off the stack.

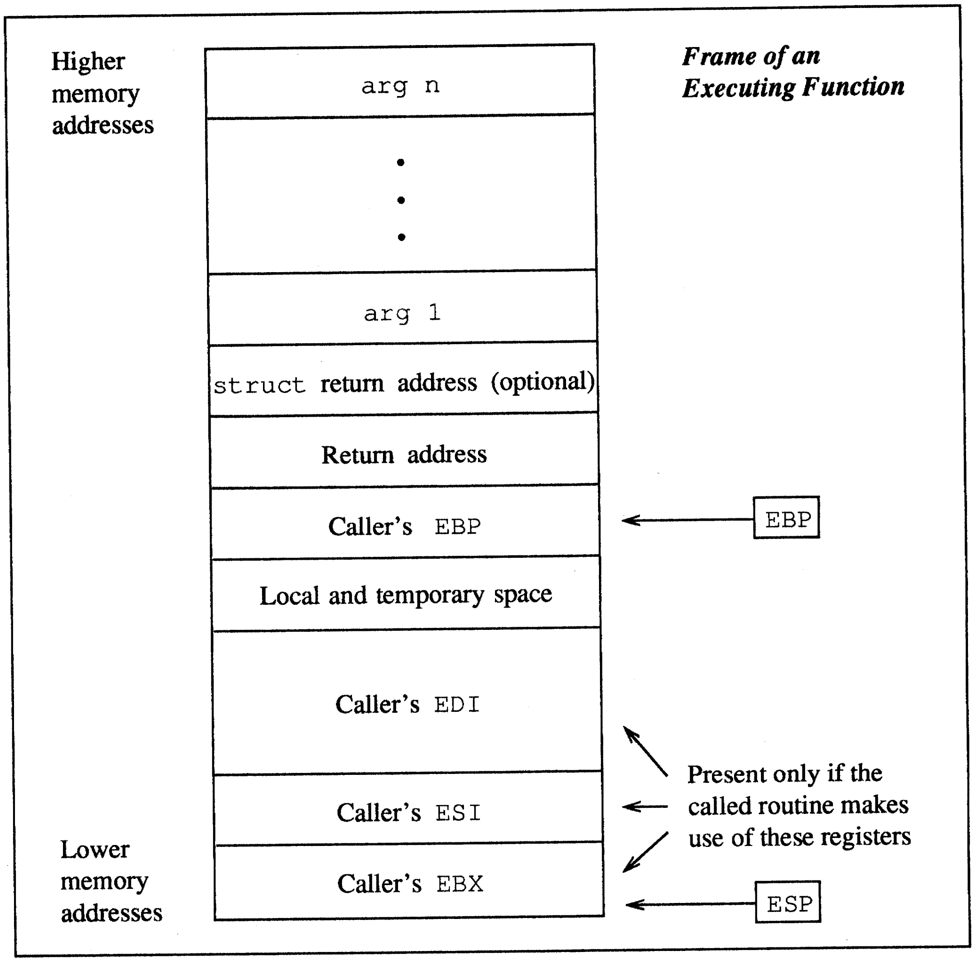

Higher memory addresses
arg n
•
•
•
arg 1
struct return address (optional)
Return address
Caller's EBP
Local and temporary space
Caller's EDI
Caller's ESI
Caller's EBX
Lower memory addresses
Frame of an Executing Function
EBP
Present only if the called routine makes use of these registers
ESP

Figure F-1 *Stack Frame for Function Invocation*

G

man Page Differences for the Sun386i System

G

man Page Differences for the Sun386i System

This appendix is divided into three areas:

- man pages that pertain only to the Sun386i system
- man pages that were altered to include Sun386i information
- SunOS 4.0 man pages that do not pertain to the Sun386i system

G.1. New man Pages for the Sun386i System

This section lists the new man commands that pertain only to the Sun386i system, divided according to their numbered section.

man(1) Commands

man(1) commands are publicly accessible user commands.

bar(1) — create tape archives, and add or extract files

cluster(1) — find the Application SunOS or Developer's Toolkit cluster containing a file

coloredit(1) — alter colormap segment

dis(1) — object code disassembler for COFF

dos(1) — SunView window for IBM PC/AT applications

dos2unix(1) — convert text file from DOS format to SunOS format

help_viewer(1) — SunView application providing help with applications and desktop

load, loadc(1) — load Application SunOS or Developer's Toolkit clusters

objdump(1) — dump selected parts of a COFF object file

organizer(1) — file and directory manager

snap(1) — SunView application for system and network administration

sysex(1) — invoke the system exerciser

syswait(1) — execute a command string, suspending termination until user input

unix2dos(1) — convert text file from SunOS format to DOS format

unload, unloadc(1) — unload Application SunOS or Developer's Toolkit clusters

man(3) Commands

man(3) commands are user-level C library functions.

ldfcn(3) — common object file access routines

man(3R) Commands

man(3R) commands are part of the RPC (Remote Procedure Call) services library.

ipalloc(3R) — determine or temporarily allocate IP address
pnp(3R) — automatic system installation

man(3X) Commands

man(3X) commands are miscellaneous functions.

ldahread(3X) — read the archive header of a member of a COFF archive file
ldclose(3X) — close a COFF file
ldfhread(3X) — read the file header of a COFF file
ldgetname(3X) — retrieve symbol name for COFF file symbol table entry
ldlread(3X) — manipulate line number entries of a COFF file function
ldlseek(3X) — seek to line number entries of a section of a COFF file
ldohseek(3X) — seek to the optional file header of a COFF file
ldopen(3X) — open a COFF file for reading
ldrseek(3X) — seek to relocation entries of section of a COFF file
ldshread(3X) — read an indexed/named section header of a COFF file
ldsseek(3X) — seek to an indexed/named section of a COFF file
ldtbindex(3X) — compute the index of a symbol table entry of a COFF file
ldtbread(3X) — read an indexed symbol table entry of a COFF file
ldtbseek(3X) — seek to the symbol table of a COFF file

man(4) Descriptions

man(4) pages describe device drivers, protocols, and network interfaces.

pp(4) — Centronics-compatible parallel printer port

man(4S) Descriptions

man(4S) pages describe devices and network interfaces.

cgthree(4S) — Sun386i color memory frame buffer
fd(4S) — disk driver for diskette controllers
root(4S) — pseudo-driver for Sun root disk

man(5) Descriptions

man(5) descriptions explain file formats.

auto.home(5) — automount map for home directories
auto.vol(5) — automount map for volumes
bar(5) — tape archive file format
coff(5) — common assembler and link editor output
group(5) — group file
help(5) — help file format
help_viewer(5) — help viewer file format
internat(5) — key mapping table for internationalization
ipalloc.netrange(5) — range of addresses to allocate

policies(5) — network administration policies

.rgb(5) — available colors (by name) for coloredit

toc(5) — table of contents of optional clusters in Application SunOS and Developer's Toolkit

translate(5) — input and output files for system message translation

man(8) Commands

man(8) commands are general system maintenance and operation commands.

client(8) — add or remove diskless system

fdformat(8) — format a diskette

logintool(8) — graphic log-in interface

modload(8) — load a loadable module

modstat(8) — display status of loadable modules

modunload(8) — unload a loadable module

unconfigure(8) — reset the network configuration for a system

man(8C) Commands

man(8C) commands are maintenance commands.

ipallocd(8C) — Ethernet-to-IP address allocator

netconfig(8C) — automatic system installation boot service

pnpboot(8C) — automatic system installation diskless boot service

pnpd(8C) — automatic system installation daemon

G.2. man Pages Altered for the Sun386i System

The man commands listed in this section were modified to include Sun386i information. Commands are divided according to their numbered section.

man(1) Commands

man(1) commands are publicly accessible user commands.

adb(1) — general purpose debugger

ar(1) — create library archives, and add or extract files

as(1) — Sun-1, Sun-2, Sun-3, Sun-4, and Sun386i assemblers

calendar(1) — a simple reminder service

cc(1) — C compiler

click(1) — enable or disable the keyboard's keystroke click

cpp(1) — the C language preprocessor

csh(1) — a shell (command interpreter) with a C-like syntax and advanced interactive features

dbx(1) — source-level debugger

ld(1) — link editor, dynamic link editor

machid(1) — return a true exit status if the processor is of the indicated type

nm(1) — print name list

passwd(1) — change password file information

roffbib(1) — format and print a bibliographic database

setkeys(1) — modify interpretation of the keyboard

size(1) — display the size of an object file

symorder(1) — rearrange a list of symbols

man(2) Commands

man(2) commands are system calls and error numbers.

ptrace(2) — process trace
syscall(2) — indirect system call

man(3) Commands

man(3) commands are user-level C library functions.

monitor(3) — prepare an execution profile
nlist(3) — get entries from name list

man(3N) Commands

man(3N) commands are network functions.

byteorder(3N) — convert values between host and network byte order
inet(3N) — Internet address manipulation

man(4F) Descriptions

man(4F) pages describe protocol families.

inet(4F) — Internet protocol family

man(4S) Descriptions

man(4S) pages describe devices and network interfaces.

bwtwo(4S) — Sun-3/Sun-2 black and white frame buffer
dkio(4S) — generic disk control operations
fbio(4S) — general properties of frame buffers
ie(4S) — Intel 10 Mb/s Ethernet interface
mem(4S) — main memory and bus I/O space
sd(4S) — disk driver for SCSI disk controllers
st(4S) — driver for Sysgen SC 4000 (Archive) and the Emulex MT-02 tape controller
zs(4S) — Zilog 8530 SCC serial communications driver

man(5) Descriptions

man(5) descriptions explain file formats.

ar(5) — archive (library) file format
core(5) — format of a memory-image file
printcap(5) — printer capability database
syslog.conf(5) — configuration file for syslogd system log daemon

man(8) Commands

man(8) commands are general system maintenance and operation commands.

config(8) — build system configuration files
dump(8) — incremental file system dump
getty(8) — set terminal mode
rc(8) — command scripts for auto-reboot and daemons
suninstall(8) — install and upgrade the Sun Operating System
syslogd(8) — log system messages

man(8C) Commands

man(8C) commands are maintenance commands.

rarpd(8C) — DARPA Reverse Address Resolution Protocol service
tftpd(8C) — DARPA Trivial File Transfer Protocol server
ypupdated (8C) — server for changing Yellow Pages information

man(8S) Commands

man(8S) commands are maintenance commands.

boot(8S) — start the system kernel, or a standalone program
kadb(8S) — adb-like kernel and standalone-program debugger
monitor(8S) — system ROM monitor

G.3. 4.0 man Pages Not Relevant to the Sun386i System

The 4.0 man pages that do not pertain to the Sun386i system are listed below, according to their numbered section.

man(1) Commands

man(1) commands are publicly accessible user commands.

adjacentscreens(1) — connect multiple screens to SunView window driver
switcher(1) — switch attention between multiple SunView desktops on the same physical screen
tcov(1) — construct test coverage analysis and statement-by-statement profile
vgrind(1) — grind nice program listings

man(4S) Descriptions

man(4S) pages describe devices and network interfaces.

ar(4S) — archive 1/4-inch streaming tape drive

man(5) Descriptions

man(5) descriptions explain file formats.

a.out(5) — assembler and link editor output format

man(6) Commands

man(6) commands are for on-line games.

chess(6) — the game of chess
chesstool(6) — window-based front-end to chess program

man(8) Commands

man(8) commands are general system maintenance and operation commands.

mc68881version(8) — print the MC68881 mask number and approximate clock rate

H

MS-DOS and ISO Character Conversion Tables

H

MS-DOS and ISO Character Conversion Tables

This appendix shows the character mapping used when converting between ISO and MS-DOS file formats with `unix2dos -iso` and `dos2unix -iso` commands. The appendix also lists the ISO character chart and the MS-DOS character chart for reference.

Table H-1 shows the MS-DOS character set, listed by decimal value. Table H-2 shows the mapping that occurs when you issue the `dos2unix -iso` command, to correctly view text files containing MS-DOS international characters in a SunOS Text Editor window. In many cases, a character in one format is the same character in the other format; when this isn't possible, the closest approximation is made. Blank spaces indicate that the character, when translated, is converted to a space.

You can use Table H-2 in conjunction with Table H-4, which shows mapping that occurs in the opposite direction, from ISO to MS-DOS files. When you issue the `unix2dos -iso` command, the conversions shown in Table H-4 are used. Note that in some cases converting a character from MS-DOS to ISO results in a space, but converting that same character back to MS-DOS format returns the original MS-DOS character to its visible form. This occurs primarily with MS-DOS graphics characters.

Table H-3 shows the ISO character set (Table 10-2 on page 154 shows a slightly different representation of the same information).

Table H-1 *MS-DOS Character Set*

Dec	Hex	DOS	Dec	Hex	DOS	Dec	Hex	DOS
0	0H	^@	42	2aH	*	84	54H	T
1	1H	^A	43	2bH	+	85	55H	U
2	2H	^B	44	2cH	,	86	56H	V
3	3H	^C	45	2dH	-	87	57H	W
4	4H	^D	46	2eH	.	88	58H	X
5	5H	^E	47	2fH	/	89	59H	Y
6	6H	^F	48	30H	0	90	5aH	Z
7	7H	^G	49	31H	1	91	5bH	[
8	8H	^H	50	32H	2	92	5cH	\
9	9H		51	33H	3	93	5dH	]
10	aH		52	34H	4	94	5eH	^
11	bH	^K	53	35H	5	95	5fH	_
12	cH	^L	54	36H	6	96	60H	`
13	dH	^M	55	37H	7	97	61H	a
14	eH	^N	56	38H	8	98	62H	b
15	fH	^O	57	39H	9	99	63H	c
16	10H	^P	58	3aH	:	100	64H	d
17	11H	^Q	59	3bH	;	101	65H	e
18	12H	^R	60	3cH	<	102	66H	f
19	13H	^S	61	3dH	=	103	67H	g
20	14H	^T	62	3eH	>	104	68H	h
21	15H	^U	63	3fH	?	105	69H	i
22	16H	^V	64	40H	@	106	6aH	j
23	17H	^W	65	41H	A	107	6bH	k
24	18H	^X	66	42H	B	108	6cH	l
25	19H	^Y	67	43H	C	109	6dH	m
26	1aH	^Z	68	44H	D	110	6eH	n
27	1bH	^[	69	45H	E	111	6fH	o
28	1cH	^\	70	46H	F	112	70H	p
29	1dH	^]	71	47H	G	113	71H	q
30	1eH	^^	72	48H	H	114	72H	r
31	1fH	^_	73	49H	I	115	73H	s
32	20H		74	4aH	J	116	74H	t
33	21H	!	75	4bH	K	117	75H	u
34	22H	"	76	4cH	L	118	76H	v
35	23H	#	77	4dH	M	119	77H	w
36	24H	$	78	4eH	N	120	78H	x
37	25H	%	79	4fH	O	121	79H	y
38	26H	&	80	50H	P	122	7aH	z
39	27H	'	81	51H	Q	123	7bH	{
40	28H	(	82	52H	R	124	7cH	¦
41	29H	)	83	53H	S	125	7dH	}

Table H-1 *MS-DOS Character Set (continued)*

Dec	Hex	DOS	Dec	Hex	DOS	Dec	Hex	DOS
126	7eH	~	169	a9H	⌐	212	d4H	╘
127	7fH	^?	170	aaH	¬	213	d5H	╒
128	80H	Ç	171	abH	½	214	d6H	╓
129	81H	ü	172	acH	¼	215	d7H	╫
130	82H	é	173	adH	¡	216	d8H	╪
131	83H	â	174	aeH	«	217	d9H	┘
132	84H	ä	175	afH	»	218	daH	┌
133	85H	à	176	b0H	░	219	dbH	█
134	86H	å	177	b1H	▒	220	dcH	▄
135	87H	ç	178	b2H	▓	221	ddH	▌
136	88H	ê	179	b3H	│	222	deH	▐
137	89H	ë	180	b4H	┤	223	dfH	▀
138	8aH	è	181	b5H	╡	224	e0H	α
139	8bH	ï	182	b6H	╢	225	e1H	β
140	8cH	î	183	b7H	╖	226	e2H	Γ
141	8dH	ì	184	b8H	╕	227	e3H	π
142	8eH	Ä	185	b9H	╣	228	e4H	Σ
143	8fH	Å	186	baH	║	229	e5H	σ
144	90H	É	187	bbH	╗	230	e6H	µ
145	91H	æ	188	bcH	╝	231	e7H	τ
146	92H	Æ	189	bdH	╜	232	e8H	Φ
147	93H	ô	190	beH	╛	233	e9H	Θ
148	94H	ö	191	bfH	┐	234	eaH	Ω
149	95H	ò	192	c0H	└	235	ebH	δ
150	96H	û	193	c1H	┴	236	ecH	∞
151	97H	ù	194	c2H	┬	237	edH	φ
152	98H	ÿ	195	c3H	├	238	eeH	ε
153	99H	Ö	196	c4H	─	239	efH	∩
154	9aH	Ü	197	c5H	┼	240	f0H	≡
155	9bH	¢	198	c6H	╞	241	f1H	±
156	9cH	£	199	c7H	╟	242	f2H	≥
157	9dH	¥	200	c8H	╚	243	f3H	≤
158	9eH	₧	201	c9H	╔	244	f4H	⌠
159	9fH	ƒ	202	caH	╩	245	f5H	⌡
160	a0H	á	203	cbH	╦	246	f6H	÷
161	a1H	í	204	ccH	╠	247	f7H	≈
162	a2H	ó	205	cdH	═	248	f8H	°
163	a3H	ú	206	ceH	╬	249	f9H	∙
164	a4H	ñ	207	cfH	╧	250	faH	·
165	a5H	Ñ	208	d0H	╨	251	fbH	√
166	a6H	ª	209	d1H	╤	252	fcH	ⁿ
167	a7H	º	210	d2H	╥	253	fdH	²
168	a8H	¿	211	d3H	╙	254	feH	■
						255	ffH	^?

Table H-2 *MS-DOS to ISO Conversion*

MS-DOS		ISO			MS-DOS		ISO		
Dec	Hex	Dec	Hex	ISO	Dec	Hex	Dec	Hex	ISO
0	0H	0	0H	^@	42	2aH	42	2aH	*
1	1H	1	1H	^A	43	2bH	43	2bH	+
2	2H	2	2H	^B	44	2cH	44	2cH	,
3	3H	3	3H	^C	45	2dH	45	2dH	-
4	4H	4	4H	^D	46	2eH	46	2eH	.
5	5H	5	5H	^E	47	2fH	47	2fH	/
6	6H	6	6H	^F	48	30H	48	30H	0
7	7H	7	7H	^G	49	31H	49	31H	1
8	8H	8	8H	^H	50	32H	50	32H	2
9	9H	9	9H		51	33H	51	33H	3
10	aH	10	aH		52	34H	52	34H	4
11	bH	11	bH	^K	53	35H	53	35H	5
12	cH	12	cH	^L	54	36H	54	36H	6
13	dH	10	aH		55	37H	55	37H	7
14	eH	14	eH	^N	56	38H	56	38H	8
15	fH	15	fH	^O	57	39H	57	39H	9
16	10H	16	10H	^P	58	3aH	58	3aH	:
17	11H	17	11H	^Q	59	3bH	59	3bH	;
18	12H	18	12H	^R	60	3cH	60	3cH	<
19	13H	19	13H	^S	61	3dH	61	3dH	=
20	14H	20	14H	^T	62	3eH	62	3eH	>
21	15H	21	15H	^U	63	3fH	63	3fH	?
22	16H	22	16H	^V	64	40H	64	40H	@
23	17H	23	17H	^W	65	41H	65	41H	A
24	18H	24	18H	^X	66	42H	66	42H	B
25	19H	25	19H	^Y	67	43H	67	43H	C
26	1aH	26	1aH	^Z	68	44H	68	44H	D
27	1bH	27	1bH	^[	69	45H	69	45H	E
28	1cH	28	1cH	^\	70	46H	70	46H	F
29	1dH	29	1dH	^]	71	47H	71	47H	G
30	1eH	30	1eH	^^	72	48H	72	48H	H
31	1fH	31	1fH	^_	73	49H	73	49H	I
32	20H	32	20H		74	4aH	74	4aH	J
33	21H	33	21H	!	75	4bH	75	4bH	K
34	22H	34	22H	"	76	4cH	76	4cH	L
35	23H	35	23H	#	77	4dH	77	4dH	M
36	24H	36	24H	$	78	4eH	78	4eH	N
37	25H	37	25H	%	79	4fH	79	4fH	O
38	26H	38	26H	&	80	50H	80	50H	P
39	27H	39	27H	'	81	51H	81	51H	Q
40	28H	40	28H	(	82	52H	82	52H	R
41	29H	41	29H	)	83	53H	83	53H	S

Table H-2 *MS-DOS to ISO Conversion (continued)*

MS-DOS		ISO			MS-DOS		ISO		
Dec	Hex	Dec	Hex	ISO	Dec	Hex	Dec	Hex	ISO
84	54H	84	54H	T	126	7eH	126	7eH	~
85	55H	85	55H	U	127	7fH	127	7fH	^?
86	56H	86	56H	V	128	80H	199	c7H	Ç
87	57H	87	57H	W	129	81H	252	fcH	ü
88	58H	88	58H	X	130	82H	233	e9H	é
89	59H	89	59H	Y	131	83H	226	e2H	â
90	5aH	90	5aH	Z	132	84H	228	e4H	ä
91	5bH	91	5bH	[	133	85H	224	e0H	à
92	5cH	92	5cH	\	134	86H	229	e5H	å
93	5dH	93	5dH	]	135	87H	231	e7H	ç
94	5eH	94	5eH	^	136	88H	234	eaH	ê
95	5fH	95	5fH	_	137	89H	235	ebH	ë
96	60H	96	60H	`	138	8aH	232	e8H	è
97	61H	97	61H	a	139	8bH	239	efH	ï
98	62H	98	62H	b	140	8cH	238	eeH	î
99	63H	99	63H	c	141	8dH	236	ecH	ì
100	64H	100	64H	d	142	8eH	196	c4H	Ä
101	65H	101	65H	e	143	8fH	197	c5H	Å
102	66H	102	66H	f	144	90H	201	c9H	É
103	67H	103	67H	g	145	91H	230	e6H	æ
104	68H	104	68H	h	146	92H	198	c6H	Æ
105	69H	105	69H	i	147	93H	244	f4H	ô
106	6aH	106	6aH	j	148	94H	246	f6H	ö
107	6bH	107	6bH	k	149	95H	242	f2H	ò
108	6cH	108	6cH	l	150	96H	251	fbH	û
109	6dH	109	6dH	m	151	97H	249	f9H	ù
110	6eH	110	6eH	n	152	98H	255	ffH	ÿ
111	6fH	111	6fH	o	153	99H	214	d6H	Ö
112	70H	112	70H	p	154	9aH	220	dcH	Ü
113	71H	113	71H	q	155	9bH	162	a2H	¢
114	72H	114	72H	r	156	9cH	163	a3H	£
115	73H	115	73H	s	157	9dH	165	a5H	¥
116	74H	116	74H	t	158	9eH	32	20H	
117	75H	117	75H	u	159	9fH	32	20H	
118	76H	118	76H	v	160	a0H	225	e1H	á
119	77H	119	77H	w	161	a1H	237	edH	í
120	78H	120	78H	x	162	a2H	243	f3H	ó
121	79H	121	79H	y	163	a3H	250	faH	ú
122	7aH	122	7aH	z	164	a4H	241	f1H	ñ
123	7bH	123	7bH	{	165	a5H	209	d1H	Ñ
124	7cH	124	7cH	|	166	a6H	170	aaH	ª
125	7dH	125	7dH	}	167	a7H	186	baH	º

Table H-2 *MS-DOS to ISO Conversion (continued)*

MS-DOS		ISO			MS-DOS		ISO		
Dec	Hex	Dec	Hex	ISO	Dec	Hex	Dec	Hex	ISO
168	a8H	191	bfH	¿	212	d4H	32	20H	
169	a9H	172	acH	¬	213	d5H	32	20H	
170	aaH	32	20H		214	d6H	32	20H	
171	abH	189	bdH	½	215	d7H	32	20H	
172	acH	188	bcH	¼	216	d8H	32	20H	
173	adH	161	a1H	¡	217	d9H	32	20H	
174	aeH	171	abH	«	218	daH	32	20H	
175	afH	187	bbH	»	219	dbH	32	20H	
176	b0H	32	20H		220	dcH	32	20H	
177	b1H	32	20H		221	ddH	32	20H	
178	b2H	32	20H		222	deH	32	20H	
179	b3H	32	20H		223	dfH	32	20H	
180	b4H	32	20H		224	e0H	32	20H	
181	b5H	32	20H		225	e1H	223	dfH	ß
182	b6H	32	20H		226	e2H	32	20H	
183	b7H	32	20H		227	e3H	32	20H	
184	b8H	32	20H		228	e4H	32	20H	
185	b9H	32	20H		229	e5H	32	20H	
186	baH	32	20H		230	e6H	181	b5H	µ
187	bbH	32	20H		231	e7H	32	20H	
188	bcH	32	20H		232	e8H	222	deH	Þ
189	bdH	32	20H		233	e9H	32	20H	
190	beH	32	20H		234	eaH	32	20H	
191	bfH	32	20H		235	ebH	240	f0H	ð
192	c0H	32	20H		236	ecH	32	20H	
193	c1H	32	20H		237	edH	248	f8H	ø
194	c2H	32	20H		238	eeH	32	20H	
195	c3H	32	20H		239	efH	32	20H	
196	c4H	32	20H		240	f0H	32	20H	
197	c5H	32	20H		241	f1H	177	b1H	±
198	c6H	32	20H		242	f2H	32	20H	
199	c7H	32	20H		243	f3H	32	20H	
200	c8H	32	20H		244	f4H	32	20H	
201	c9H	32	20H		245	f5H	32	20H	
202	caH	32	20H		246	f6H	247	f7H	÷
203	cbH	32	20H		247	f7H	32	20H	
204	ccH	32	20H		248	f8H	32	20H	
205	cdH	32	20H		249	f9H	32	20H	
206	ceH	32	20H		250	faH	32	20H	
207	cfH	32	20H		251	fbH	32	20H	
208	d0H	32	20H		252	fcH	32	20H	
209	d1H	32	20H		253	fdH	178	b2H	²
210	d2H	32	20H		254	feH	32	20H	
211	d3H	32	20H		255	ffH	32	20H	

Table H-3 *ISO Character Set*

Dec	Hex	ISO	Dec	Hex	ISO	Dec	Hex	ISO
0	0H	^@	42	2aH	*	84	54H	T
1	1H	^A	43	2bH	+	85	55H	U
2	2H	^B	44	2cH	,	86	56H	V
3	3H	^C	45	2dH	-	87	57H	W
4	4H	^D	46	2eH	.	88	58H	X
5	5H	^E	47	2fH	/	89	59H	Y
6	6H	^F	48	30H	0	90	5aH	Z
7	7H	^G	49	31H	1	91	5bH	[
8	8H	^H	50	32H	2	92	5cH	\
9	9H		51	33H	3	93	5dH	]
10	aH		52	34H	4	94	5eH	^
11	bH	^K	53	35H	5	95	5fH	_
12	cH	^L	54	36H	6	96	60H	`
13	dH	^M	55	37H	7	97	61H	a
14	eH	^N	56	38H	8	98	62H	b
15	fH	^O	57	39H	9	99	63H	c
16	10H	^P	58	3aH	:	100	64H	d
17	11H	^Q	59	3bH	;	101	65H	e
18	12H	^R	60	3cH	<	102	66H	f
19	13H	^S	61	3dH	=	103	67H	g
20	14H	^T	62	3eH	>	104	68H	h
21	15H	^U	63	3fH	?	105	69H	i
22	16H	^V	64	40H	@	106	6aH	j
23	17H	^W	65	41H	A	107	6bH	k
24	18H	^X	66	42H	B	108	6cH	l
25	19H	^Y	67	43H	C	109	6dH	m
26	1aH	^Z	68	44H	D	110	6eH	n
27	1bH	^[	69	45H	E	111	6fH	o
28	1cH	^\	70	46H	F	112	70H	p
29	1dH	^]	71	47H	G	113	71H	q
30	1eH	^^	72	48H	H	114	72H	r
31	1fH	^_	73	49H	I	115	73H	s
32	20H		74	4aH	J	116	74H	t
33	21H	!	75	4bH	K	117	75H	u
34	22H	"	76	4cH	L	118	76H	v
35	23H	#	77	4dH	M	119	77H	w
36	24H	$	78	4eH	N	120	78H	x
37	25H	%	79	4fH	O	121	79H	y
38	26H	&	80	50H	P	122	7aH	z
39	27H	'	81	51H	Q	123	7bH	{
40	28H	(	82	52H	R	124	7cH	|
41	29H	)	83	53H	S	125	7dH	}

Table H-3 *ISO Character Set (continued)*

Dec	Hex	ISO	Dec	Hex	ISO	Dec	Hex	ISO
126	7eH	~	169	a9H	©	212	d4H	Ô
127	7fH	^?	170	aaH	ª	213	d5H	Õ
128	80H		171	abH	«	214	d6H	Ö
129	81H		172	acH	¬	215	d7H	×
130	82H		173	adH	^?	216	d8H	Ø
131	83H		174	aeH	®	217	d9H	Ù
132	84H		175	afH	¯	218	daH	Ú
133	85H		176	b0H	°	219	dbH	Û
134	86H	^?	177	b1H	±	220	dcH	Ü
135	87H	^?	178	b2H	²	221	ddH	Ý
136	88H	^?	179	b3H	³	222	deH	Þ
137	89H	^?	180	b4H	´	223	dfH	ß
138	8aH	^?	181	b5H	µ	224	e0H	à
139	8bH	^?	182	b6H	¶	225	e1H	á
140	8cH	^?	183	b7H	•	226	e2H	â
141	8dH	^?	184	b8H	¸	227	e3H	ã
142	8eH	^?	185	b9H	¹	228	e4H	ä
143	8fH	^?	186	baH	º	229	e5H	å
144	90H	^?	187	bbH	»	230	e6H	æ
145	91H	^?	188	bcH	¼	231	e7H	ç
146	92H	^?	189	bdH	½	232	e8H	è
147	93H	^?	190	beH	¾	233	e9H	é
148	94H	^?	191	bfH	¿	234	eaH	ê
149	95H	^?	192	c0H	À	235	ebH	ë
150	96H	^?	193	c1H	Á	236	ecH	ì
151	97H	^?	194	c2H	Â	237	edH	í
152	98H	^?	195	c3H	Ã	238	eeH	î
153	99H	^?	196	c4H	Ä	239	efH	ï
154	9aH	^?	197	c5H	Å	240	f0H	ð
155	9bH	^?	198	c6H	Æ	241	f1H	ñ
156	9cH	^?	199	c7H	Ç	242	f2H	ò
157	9dH	^?	200	c8H	È	243	f3H	ó
158	9eH	^?	201	c9H	É	244	f4H	ô
159	9fH	^?	202	caH	Ê	245	f5H	õ
160	a0H	^?	203	cbH	Ë	246	f6H	ö
161	a1H	¡	204	ccH	Ì	247	f7H	÷
162	a2H	¢	205	cdH	Í	248	f8H	ø
163	a3H	£	206	ceH	Î	249	f9H	ù
164	a4H	¤	207	cfH	Ï	250	faH	ú
165	a5H	¥	208	d0H	Ð	251	fbH	û
166	a6H	¦	209	d1H	Ñ	252	fcH	ü
167	a7H	§	210	d2H	Ò	253	fdH	ý
168	a8H	¨	211	d3H	Ó	254	feH	þ
						255	ffH	ÿ

Table H-4 *ISO to MS-DOS Conversion*

ISO		MS-DOS			ISO		MS-DOS		
Dec	Hex	Dec	Hex	DOS	Dec	Hex	Dec	Hex	DOS
0	0H	0	0H	^@	42	2aH	42	2aH	*
1	1H	1	1H	^A	43	2bH	43	2bH	+
2	2H	2	2H	^B	44	2cH	44	2cH	,
3	3H	3	3H	^C	45	2dH	45	2dH	-
4	4H	4	4H	^D	46	2eH	46	2eH	.
5	5H	5	5H	^E	47	2fH	47	2fH	/
6	6H	6	6H	^F	48	30H	48	30H	0
7	7H	7	7H	^G	49	31H	49	31H	1
8	8H	8	8H	^H	50	32H	50	32H	2
9	9H	9	9H		51	33H	51	33H	3
10	aH	10	aH		52	34H	52	34H	4
11	bH	11	bH	^K	53	35H	53	35H	5
12	cH	12	cH	^L	54	36H	54	36H	6
13	dH	13	dH	^M	55	37H	55	37H	7
14	eH	14	eH	^N	56	38H	56	38H	8
15	fH	15	fH	^O	57	39H	57	39H	9
16	10H	16	10H	^P	58	3aH	58	3aH	:
17	11H	17	11H	^Q	59	3bH	59	3bH	;
18	12H	18	12H	^R	60	3cH	60	3cH	<
19	13H	19	13H	^S	61	3dH	61	3dH	=
20	14H	20	14H	^T	62	3eH	62	3eH	>
21	15H	21	15H	^U	63	3fH	63	3fH	?
22	16H	22	16H	^V	64	40H	64	40H	@
23	17H	23	17H	^W	65	41H	65	41H	A
24	18H	24	18H	^X	66	42H	66	42H	B
25	19H	25	19H	^Y	67	43H	67	43H	C
26	1aH	26	1aH	^Z	68	44H	68	44H	D
27	1bH	27	1bH	^[	69	45H	69	45H	E
28	1cH	28	1cH	^\	70	46H	70	46H	F
29	1dH	29	1dH	^]	71	47H	71	47H	G
30	1eH	30	1eH	^^	72	48H	72	48H	H
31	1fH	31	1fH	^_	73	49H	73	49H	I
32	20H	32	20H		74	4aH	74	4aH	J
33	21H	33	21H	!	75	4bH	75	4bH	K
34	22H	34	22H	"	76	4cH	76	4cH	L
35	23H	35	23H	#	77	4dH	77	4dH	M
36	24H	36	24H	$	78	4eH	78	4eH	N
37	25H	37	25H	%	79	4fH	79	4fH	O
38	26H	38	26H	&	80	50H	80	50H	P
39	27H	39	27H	'	81	51H	81	51H	Q
40	28H	40	28H	(	82	52H	82	52H	R
41	29H	41	29H	)	83	53H	83	53H	S

Table H-4 *ISO to MS-DOS Conversion (continued)*

ISO		MS-DOS			ISO		MS-DOS		
Dec	Hex	Dec	Hex	DOS	Dec	Hex	Dec	Hex	DOS
84	54H	84	54H	T	126	7eH	126	7eH	~
85	55H	85	55H	U	127	7fH	127	7fH	^?
86	56H	86	56H	V	128	80H	32	20H	
87	57H	87	57H	W	129	81H	32	20H	
88	58H	88	58H	X	130	82H	32	20H	
89	59H	89	59H	Y	131	83H	32	20H	
90	5aH	90	5aH	Z	132	84H	32	20H	
91	5bH	91	5bH	[	133	85H	32	20H	
92	5cH	92	5cH	\	134	86H	32	20H	
93	5dH	93	5dH	]	135	87H	32	20H	
94	5eH	94	5eH	^	136	88H	32	20H	
95	5fH	95	5fH	_	137	89H	32	20H	
96	60H	96	60H	`	138	8aH	32	20H	
97	61H	97	61H	a	139	8bH	32	20H	
98	62H	98	62H	b	140	8cH	32	20H	
99	63H	99	63H	c	141	8dH	32	20H	
100	64H	100	64H	d	142	8eH	32	20H	
101	65H	101	65H	e	143	8fH	32	20H	
102	66H	102	66H	f	144	90H	32	20H	
103	67H	103	67H	g	145	91H	32	20H	
104	68H	104	68H	h	146	92H	32	20H	
105	69H	105	69H	i	147	93H	32	20H	
106	6aH	106	6aH	j	148	94H	32	20H	
107	6bH	107	6bH	k	149	95H	32	20H	
108	6cH	108	6cH	l	150	96H	32	20H	
109	6dH	109	6dH	m	151	97H	32	20H	
110	6eH	110	6eH	n	152	98H	32	20H	
111	6fH	111	6fH	o	153	99H	32	20H	
112	70H	112	70H	p	154	9aH	32	20H	
113	71H	113	71H	q	155	9bH	32	20H	
114	72H	114	72H	r	156	9cH	32	20H	
115	73H	115	73H	s	157	9dH	32	20H	
116	74H	116	74H	t	158	9eH	32	20H	
117	75H	117	75H	u	159	9fH	32	20H	
118	76H	118	76H	v	160	a0H	32	20H	
119	77H	119	77H	w	161	a1H	173	adH	¡
120	78H	120	78H	x	162	a2H	155	9bH	¢
121	79H	121	79H	y	163	a3H	156	9cH	£
122	7aH	122	7aH	z	164	a4H	32	20H	
123	7bH	123	7bH	{	165	a5H	157	9dH	¥
124	7cH	124	7cH	|	166	a6H	32	20H	
125	7dH	125	7dH	}	167	a7H	32	20H	
					168	a8H	32	20H	

Table H-4 *ISO to MS-DOS Conversion (continued)*

ISO		MS-DOS			ISO		MS-DOS		
Dec	Hex	Dec	Hex	DOS	Dec	Hex	Dec	Hex	DOS
169	a9H	32	20H		212	d4H	147	93H	ô
170	aaH	172	acH	¼	213	d5H	111	6fH	o
171	abH	174	aeH	«	214	d6H	153	99H	Ö
172	acH	191	bfH	┐	215	d7H	32	20H	
173	adH	32	20H		216	d8H	237	edH	φ
174	aeH	32	20H		217	d9H	151	97H	ù
175	afH	32	20H		218	daH	163	a3H	ú
176	b0H	248	f8H	°	219	dbH	150	96H	û
177	b1H	241	f1H	±	220	dcH	154	9aH	Ü
178	b2H	253	fdH	²	221	ddH	89	59H	Y
179	b3H	51	33H	3	222	deH	232	e8H	Φ
180	b4H	39	27H	´	223	dfH	225	e1H	β
181	b5H	236	ecH	∞	224	e0H	133	85H	à
182	b6H	20	14H	^T	225	e1H	160	a0H	á
183	b7H	32	20H		226	e2H	131	83H	â
184	b8H	32	20H		227	e3H	97	61H	a
185	b9H	49	31H	1	228	e4H	132	84H	ä
186	baH	167	a7H	º	229	e5H	134	86H	å
187	bbH	175	afH	»	230	e6H	145	91H	æ
188	bcH	172	acH	¼	231	e7H	135	87H	ç
189	bdH	171	abH	½	232	e8H	138	8aH	è
190	beH	32	20H		233	e9H	130	82H	é
191	bfH	168	a8H	¿	234	eaH	136	88H	ê
192	c0H	133	85H	à	235	ebH	137	89H	ë
193	c1H	160	a0H	á	236	ecH	141	8dH	ì
194	c2H	131	83H	â	237	edH	161	a1H	í
195	c3H	97	61H	a	238	eeH	140	8cH	î
196	c4H	142	8eH	Ä	239	efH	139	8bH	ï
197	c5H	143	8fH	Å	240	f0H	235	ebH	δ
198	c6H	146	92H	Æ	241	f1H	164	a4H	ñ
199	c7H	128	80H	Ç	242	f2H	149	95H	ò
200	c8H	138	8aH	è	243	f3H	162	a2H	ó
201	c9H	144	90H	É	244	f4H	147	93H	ô
202	caH	136	88H	ê	245	f5H	111	6fH	o
203	cbH	137	89H	ë	246	f6H	148	94H	ö
204	ccH	141	8dH	ì	247	f7H	246	f6H	÷
205	cdH	161	a1H	í	248	f8H	237	edH	φ
206	ceH	140	8cH	î	249	f9H	151	97H	ù
207	cfH	139	8bH	ï	250	faH	163	a3H	ú
208	d0H	68	44H	D	251	fbH	150	96H	û
209	d1H	165	a5H	Ñ	252	fcH	129	81H	ü
210	d2H	149	95H	ò	253	fdH	121	79H	y
211	d3H	162	a2H	ó	254	feH	232	e8H	Φ
					255	ffH	152	98H	ÿ

Index

Notes