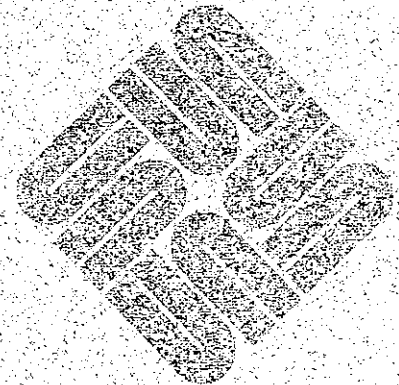




Software Technical Bulletin
February 1988

Software Information Services

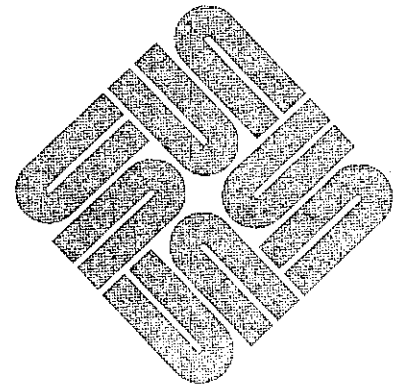






Software Technical Bulletin
February 1988

Software Information Services



Part Number 812-8801-02
Issue 1988 - 02
February 1988

Software Technical Bulletins are distributed to customers with software/hardware or software only support contracts. Send comments or corrections to 'Software Technical Bulletins' at Sun Microsystems, Inc., 2550 Garcia Ave., M/S 2-312, Mountain View, CA 94043 or by electronic mail to *sun!stb-editor*. U.S customers who have technical questions about topics in the Bulletin should call the Sun Customer Software Services AnswerLine at 800 USA-4-SUN. Other customers should call the numbers listed in *World Hotlines* appearing in Section 1.

UNIX, UNIX/32V, UNIX System III, and UNIX System V are trademarks of AT&T Bell Laboratories. DEC, DNA, VAX, VMS, VT100, WPS-PLUS, and Ultrix are registered trademarks of Digital Equipment Corporation.

Courier 2400 is a trademark of U.S. Robotics, Inc.

Hayes is a trademark of Hayes Microcomputer Products, Inc.

Multibus is a trademark of Intel Corporation.

PostScript and TranScript are trademarks of Adobe Systems, Inc.

Ven-Tel is a trademark of Ven-Tel, Inc.

Sun-2, Sun-2/xxx, Sun-3, Deskside, SunStation, Sun Workstation, SunCore, DVMA, SunWindows, NeWS, NFS, SunUNIFY™, SunView™, SunGKS, SunCGI, SunGuide, SunSimplify, SunLink, Sun Microsystems, and the Sun logo are trademarks of Sun Microsystems, Inc.

UNIFY™ is a trademark of Unify Corporation.

ENTER, PAINT, ACCELL, and RPT are trademarks of Unify Corporation.

SQL™ is a trademark of International Business Machines Corporation.

Applix® is a registered trademark of Applix, Inc.

SunAlis™ is a trademark of Sun Microsystems, Inc. and is derived from Alis, a product marketed by Applix, Inc.

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

Copyright © 1988 by Sun Microsystems.

This publication is protected by Federal Copyright Law, with all rights reserved. No part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, or by any means manual, electric, electronic, electro-magnetic, mechanical, chemical, optical, or otherwise, without prior explicit written permission from Sun Microsystems.

Contents

Section 1 NOTES & COMMENTS	193
Editor's Notes	193
Software Release Levels	195
World Hotlines	198
STB Duplication	199
Section 2 ARTICLES	203
Enhanced SCSI Adapters	203
FORTRAN Release 1.05	204
SysOS 4-3.2 Announcement	205
Creating Aliases	209
Cross Compilers	214
Section 3 STB SHORT SUBJECTS	219
Setting MAXUSERS	219
<i>Domain</i> Definitions	221
Section 4 IN DEPTH	225
Graphics Standards	225
Section 5 QUESTIONS, ANSWERS, HINTS, AND TIPS	255
Q&A, and Tip of the Month	255
Section 6 THE HACKERS' CORNER	261
Backup Copy Daemon	261
Section 7 CUMULATIVE INDEX: 1988	277



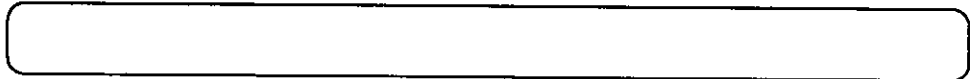
NOTES & COMMENTS

NOTES & COMMENTS	193
Editor's Notes	193
Software Release Levels	195
World Hotlines	198
STB Duplication	199



NOTES & COMMENTS

Editor's Notes



Editor's Notes

The February 1988 Software Technical Bulletin (STB) editor's notes include notes on the monthly software product release tables, world hotlines for use by customers outside the U.S., STB duplication permission, the cumulative index, and the **Hackers' Corner**.

Expanded Current Sun Software Products and Release Level Tables

The five tables showing current Sun software product release levels appear monthly. These tables show release levels for operating systems, communications products, unbundled languages, and unbundled applications.

World Hotlines

For Sun customers served by your local service groups, use the customer service telephone numbers listed in this monthly item. Also, look to this section during the upcoming year for details on your local support call policies and procedures.

STB Duplication Permission

This notice is published monthly, giving customers useful information regarding ordering and duplicating additional STB copies.

The 1988 Cumulative Index

Note that beginning with last month's January 1988 STB issue, the cumulative pagination was reset to page one. This February issue and subsequent 1988 issues will continue cumulative pagination throughout the new year.

The Hackers' Corner

This month's **Hackers' Corner** includes code that allows system administrators to conveniently make backup copies of filesystems using the `cp` or `rcp` commands.

Again, please note that such applications, scripts, or code are not offered as released Sun products, but as items of interest to enthusiasts wanting to try out something for themselves. They may not work in all cases, and may not be compatible with future SunOS releases. Please consult your local shell script or programming expert regarding any application, script, or code problems.

Thanks.

The STB Editor

Software Release Levels

As of December 18, 1987

Operating Systems

Product Name	Current Release
SunOS (Sun-2 and Sun-3 Operating System)	3.4
Sys4 (Sun-4 Operating System)	3.2

Communications Products

Product Name	Current Release
SunLink BSC3270	3.0
SunLink BSCRJE	5.0
SunLink Local 3270	5.0
SunLink SNA3270	5.0
SunLink Peer-to-Peer	5.0
SunLink IR	5.0
SunLink DDN	5.0
SunLink DNI	5.0
SunLink OSI	5.0
SunLink MCP	5.0
SunLink TE100	5.0
SunLink X.25	5.0

Unbundled Languages

Product Name	Current Release
Sun FORTRAN* (for Sun-2 and Sun-3 systems)	1.0
Sun FORTRAN* (for Sun-4 systems)	1.05
SunPro	2.0
NeWS	1.0
Sun Common Lisp-D	2.1
Sun Common Lisp-E	1.1
Modula-2	1.0
Cross Compilers	2.0

* Sun FORTRAN Note

The $\epsilon 77$ compiler is automatically included with SunOS release 3, which includes SunOS releases 3.2, 3.4, and 3.5. Sun FORTRAN release 1.0 (for Sun-2 and Sun-3 systems) and Sun FORTRAN Release 1.05 (for Sun-4 systems) are value-added products that support VMS extensions to the $\epsilon 77$ compiler, and must be purchased separately from the operating system.

Unbundled Graphics

Product Name	Current Release
SunGKS	2.1

Unbundled Applications

Product Name	Current Release
SunAlis	2.1
SunINGRES	5.0
SunSimplify	1.0
SunUNIFY	2.0
Transcript	2.0
SunIPC	1.1
PC-NFS	2.0
SunTrac (for Sun-2 and Sun-3 systems)	1.0
SunTrac (for Sun-4 systems)	1.0/3.2


**Current Sun Software
Products and Release Levels**

The preceding tables contain lists of current Sun software products and their respective current release levels.

You will note that the Software Technical Bulletin (STB) contains articles from time to time that detail technical changes in a given software product's next available release.

Please contact your sales representative if you decide that you would like to update the release level of a Sun software product you already use, or wish to purchase another product. Use the tables to determine whether your release is the current release level.

These tables appear monthly in the STB for your convenience.

World Hotlines



World Hotlines

Sun Customers throughout the world have service hotlines available for both software and hardware support questions. The service hotlines are shown below. If your country is not shown in the table, please phone your local Sun sales office.

Australia	Sun Australia Lionel Singer Group	(011-61-2) 957-2522 (011-61-2) 957-2655
Canada	Montreal Branch Ottawa Vancouver Branch Western Branch	(514) 879-1914 (613) 748-9617 (604) 641-1296 (403) 295-0150
France	Paris Sun Microsystems France SA	(33) 1 4630 2324
Germany	Munich Sun Microsystems GmbH	(49) 89/95094-321
Japan	C. Itoh Data Systems Nihon Sun	(011-81-3) 497-4676 (011-81-3) 221-7021
The Netherlands	Soest Sun Microsystems Nederland BV	(31) 2155 24888
Switzerland	Zurich Sun Microsystems Schweiz AG	(41) 1 828 9555
United Kingdom	Camberley Sun Microsystems UK Ltd	(44) 276 62111
United States	All, including Puerto Rico	1-800-USA-4-SUN
Intercon	All countries outside the USA, Europe, and northern Africa	(415) 691-6775



STB Duplication

Duplicating the STB

Your company's software support contract includes a monthly issue of the STB, which contains a quarterly, updated Customer Distributed BugsList (CDB). Each month, the copy of your STB is mailed to your company's primary contact person or department. Sites with more than one contract may receive more than one STB copy, depending on how the contracts are set up.

Your primary contact person or department may duplicate this 'master' STB copy for all Sun workstation end-users. So long as you duplicate copies and route them only internally, there are no copyright infringement problems.

This limited permission for duplication is for your convenience only, however, and does not include any duplication for resale, for distribution outside your company, or for distribution to employees of companies not having a Sun software support contract.

Direct STB Purchase

The STB is sent to the primary contact person named in all software support contracts. Sun is looking into methods by which customers holding these contracts may purchase extra copies directly.

Look to this column for an announcement regarding the purchase of extra STB copies.

Further Questions

If you have any questions, comments, or articles regarding the STB or CDB, please send your ideas and questions to *sun/stb-editor*.



ARTICLES

ARTICLES	203
Enhanced SCSI Adapters	203
FORTRAN Release 1.05	204
SysOS 4-3.2 Announcement	205
Creating Aliases	209
Cross Compilers	214

UNIVERSITY OF TORONTO LIBRARY



ARTICLES

Enhanced SCSI Adapters

Enhanced SCSI Host Adapters

SCSI mass storage options for VME systems now include an enhanced version of the Sun SCSI-to-VME host adapter. The enhanced SCSI host-adapter now supports the two features listed below.

- Disconnect/reconnect capability
- 32-bit data transfer over the VME bus

Software Compatibility

The enhanced SCSI-to-VME host adapter is supported under SunOS release 3.4 with the installation of a patch tape, part number 700-1656-01, and the document *Read This First*, part number 800-2190-01, shipped with each SCSI board.

Hardware Compatibility

Carrera boards, part number 501-1164-08 and below, and the Prism boards, part number 501-1134-05 and below, will not support 32-bit DMA over the VME bus and are therefore *not compatible* with the enhanced version of the SCSI host-adapter.

All systems using the Sun enhanced SCSI-to-VME host adapter must have the up-graded Carrera board , part number 501-1164-09 and above, and the Prism boards, part number 501-1134-06 and above.

Enhanced SCSI Adapter Part Numbers

Use the part numbers shown below in the case that you wish to order an enhanced SCSI-to-VME host adapter.

- 501-1236-XX, the SCSI-3
- 501-1217-02, the 3X2 Adapter with the SCSI-3 (desk top systems)
- 501-1170-05, the 3X2 Adapter with the SCSI-3 (desk side systems)

FORTRAN Release 1.05

Sun FORTRAN 1.05 Announcement

This article is a brief overview of VMS-compatible Sun FORTRAN release 1.05, which has been developed specifically for use with Sun-4 series Scalable Processor Architecture (SPARC) workstation systems, and supported *only* by the Sun Sys4-3.2 operating system.

Introduction

Sun FORTRAN release 1.05 is an enhanced ANSI FORTRAN 77 development system. It is Government Services Administration (GSA)-certified with VAX/VMS FORTRAN 4.0 extensions, and thus provides a development system for a significantly expanded body of FORTRAN source code. Most existing VMS FORTRAN applications can be ported to the Sun-4 workstation environment.

The Sun FORTRAN release 1.05 package consists of three components listed below.

- extensions to the `f77` compiler to support most of the VMS FORTRAN features
- extensions to the debugger to support most of these same VMS FORTRAN features
- the `f77cvt` Source-Code Converter program, used to convert most of the remaining extensions into statements that the Sun FORTRAN compiler will accept

When used together, the compiler and converter provide almost total compatibility with VMS FORTRAN.

Sun FORTRAN release 1.05 is essentially the same as Sun FORTRAN release 1.0, which was developed for Sun-2 and Sun-3 systems running Sun Operating System (SunOS) releases 3.2, 3.4, and 3.5. Please refer to the article *FORTRAN 1.0 Announcement* on page 707 of the October 1987 STB issue for further information.

Sun FORTRAN release 1.05 has been developed specifically for use with the Sun-4 workstation series. Therefore, this release corrects the data alignment problem that appeared in the beta version of `f77` in the Sun Sys4-3.2 operating system.

SysOS 4-3.2 Announcement

Sys4-3.2 Announcement

This article is a brief overview of Sys4-3.2, the first Sun Operating System (SunOS) release which is being shipped with all Sun-4/200 series Scalable Processor Architecture (SPARC) workstation systems.

Introduction

Sys4-3.2 for Sun-4/200 workstations includes all the functions and features supported in SunOS release 3.2 for Sun-2 and Sun-3 series workstations, including the C compiler, SunView, SunPro, SunCGI, SunCore, and Pixrects/Pixwin. The C compiler has been upgraded to generate optimized code, using the same optimization techniques employed by the Sun FORTRAN compiler.

In addition, Sys4-3.2 includes all SunView enhancements provided by SunOS release 3.4 (for Sun-2 and Sun-3 workstations), as well as special release 3.4 bug fixes as documented in the *Sys4-3.2 Release Manual*.

Sys4-3.2 incorporates the features listed below.

- Sun-4 SPARC/RISC architecture
- New hardware support
- Remote tape software installation
- Binary compatibility with SunOS 3.2L
- Support for Sun FORTRAN release 1.05
- Additional key unbundled software support

Each of these is discussed in the following paragraphs.

Sun-4 SPARC/RISC Architecture

Sun-4 SPARC, an acronym for Scalable Processor Architecture, is a Reduced Instruction Set (RISC) architecture implementation that emphasizes simplicity and efficiency through the following listed provisions.

- 32-bit wide instructions with word-aligned memory. Op-codes and addresses always appear in the same place to simplify hardware decoding.

- Register-intensive architecture: Instructions operate on two registers, or a register and constant; the result is placed in a third register. Memory is accessed through load and store instructions only.
- Large register windows: The processor has access to a large number of registers, which are configured into overlapping sets. This allows compilers to automatically cache values and pass parameters in registers.
- Delayed control transfer: The processor fetches the next instruction following a control transfer before it completes the transfer. Compilers can re-arrange code to place useful instructions after delayed control transfer, thus maximizing throughput.
- All instructions except loads, stores, and floating-point operations can be executed in a single machine cycle.

Sun-4 SPARC/RISC architecture is designed to support the following items.

- The C language and the UNIX Operating System
- SunView on SunWindows
- Numerical applications, using FORTRAN
- Artificial intelligence applications, using Lisp and Prolog
- Programs written in Sun Pascal

New Hardware Support

The Sun-4/200 series workstation and Sys4-3.2 support the following new hardware.

32 MB memory boards: The Sun-4/200 series features the new 32 MB memory boards.

ALM-2 asynchronous multiplexer: The Sun-4/200 series features support for the new ALM-2 asynchronous multiplexer, a 16-channel asynchronous multiplexer board (full-sized, 9U VME) with an additional connection to Centronics-compatible parallel printers. The ALM-2 allows up to sixteen terminals or other serial devices to connect to Sun-3 or Sun-4 workstations that have at least one slot available.

In addition, the Sun-4/200 series workstation supports the following peripherals.

- 575 MB Disk
- 280 MB Disk
- 6250/1600 bpi XY472 Tape

- 1/4-inch SCSI Tape
- GP+ and GB Graphics Boards
- ALM asynchronous multiplexer

Note that Sys4-3.2 does not support the following controller and tape drive.

- CPC Tapemaster 1600 bpi controller
- CDC 92181 Keystone 40 MB 9-track reel

Thus, upgrades from Sun-3 systems containing this controller or tape drive will not work.

Software Installation

Sys4-3.2 supports software installation for the following configurations listed below.

- Sun-4 standalone systems
- Sun-4 servers with Sun-4 clients

A Sun-4 server can support both Sun-3 and Sun-4 clients. This configuration requires SunOS release 3.5 for Sun-3 client installation, in addition to Sys4-3.2 for the Sun-4 server and client installation.

Due to network disk (nd) limitations, Sys4-3.2 does not support Sun-2 clients, or Sun-3 servers with Sun-4 clients.

Sys4-3.2 supports remote software installation on a standalone workstation, or on a file server which does not have a resident tape drive. Remote installation is performed by using the tape drive on another, fully installed machine, which can be either a standalone system or a Sun-4 server. This machine becomes the tape server, or *remote host*. Installation is thus performed from the Sun-4 client, or *target machine*, across the Ethernet network. Sun-3 target machine clients can also be remotely installed, using SunOS release 3.5 on the remote host.

Binary Compatibility with SunOS 3.2L

Programs which have been developed under SunOS release 3.2L are binary compatible, and will run under Sys4-3.2 without recompilation. This binary compatibility is designed to facilitate the transition between operating system releases. Thus, developers can begin application development for the Sun-4 system now and smoothly migrate to future SunOS releases.

Code developed on other Sun systems will require recompilation under Sys4-3.2 to enable execution on the Sun-4 system. Well-written, portable code should recompile without any source modification. Any data alignment dependencies, as well as any dependent compiler-specific implementations of unspecified functions (such as the order of C parameter evaluation) may require changes to source code. Most problems can be detected by using `lint -ch`, where the

-c detects unportable casts, and the -h flag performs heuristic checking.

For complete information on porting applications to the Sun-4 system, refer to *Porting Software to SPARC Systems*, part number 800-1596.

Support for Sun FORTRAN
Release 1.05

Sys4-3.2 supports Sun FORTRAN release 1.05, Sun's new VMS-compatible version of FORTRAN. Sun FORTRAN release 1.05 has been developed specifically for use with the Sun-4 workstation series. Therefore, this release corrects the data alignment problem that appeared in the beta version of f77 in the Sun Sys4-3.2 Operating System.


Refer to the article entitled *FORTRAN 1.05 Announcement* in this issue of the STB for further information.

Support for Key Unbundled
Software

In addition to SunFORTRAN release 1.05, Sys4-3.2 supports the following listed unbundled software.

- NeWS (Network Windowing System), release 1.1
- SunTrac Project Management System, release 1.0

For complete information on SunTrac release 1.0, refer to the December, 1987 issue of the *Software Technical Bulletin*.




Creating Aliases

Using `/usr/lib/aliases`

Sun workstation users having system accounts can establish their own personal aliases in the `/usr/lib/aliases` file, provided he or she has root access (via the `su` command). Personal aliases provide a convenient means to communicate with other users, as you can combine many individual aliases under one alias name.

Personal aliases can be established to perform the following functions:

- Establish distribution lists that can be used to send and receive mail among those included on the list, where the list is a part of the `/usr/lib/aliases` file
- Establish distribution lists to send mail messages to the same group of individuals, where the list is in a file separate from the `/usr/lib/aliases` file
- Receive your mail messages under several different aliases



The `/usr/lib/aliases` File

A `/usr/lib/aliases` file exists for each workstation user. A copy of this file in its default form appears as shown on the next page.

```

##
# Aliases can have any mix of upper and lower case on the left-hand side,
# but the right-hand side should be proper case (usually lower)
#
# >>>>>>>>>> The program "newaliases" will need to be run after
# >> NOTE >> this file is updated for any changes to
# >>>>>>>>>> show through to sendmail.
#
# @(#)aliases 1.1 86/07/08 SMI
##

# Following alias is required by the mail protocol, RFC 822
# Set it to the address of a HUMAN who deals with this system's mail problems.
Postmaster: root

# Alias for mailer daemon; returned messages from our MAILER-DAEMON
# should be routed to our local Postmaster.
MAILER-DAEMON: postmaster

# Aliases to handle mail to programs or files, eg news or vacation
decode: "|/usr/bin/uudecode"
nobody: /dev/null

# Sample aliases:

# Alias for distribution list, members specified here:
#staff:wnj,mosher,sam,ecc,mckusick,sklower,olson,rwh@ernie

# Alias for distribution list, members specified elsewhere:
#keyboards: :include:/usr/jfarrell/keyboards.list

# Alias for a person, so they can receive mail by several names:
#epa:eric

#####
# Local aliases below #
#####

```

Creating a Distribution List
within /usr/lib/aliases

This type of alias distribution list is particularly suitable for establishing a list of aliases that are short (such as a small group of individuals) or stable (such as group that doesn't change too often.)

To create a distribution list alias within the /usr/lib/aliases file, perform the following steps.

1. At the prompt, enter `su` to change to root status and display the `#` prompt.
2. At the `#` prompt, open the /usr/lib/aliases file, using your favorite editor (such as `vi`).

3. Open a new line at the bottom of the file, then add the distribution list alias name and the associated user aliases, using the following format:

```
<list_alias_name>: <user1@hostname>, <user2@hostname>, <user3@hostname> ...
```

You can include up to 1,000 characters (in the form of user aliases) in each distribution list.

Example: The following users (including yourself) will be included in a distribution list alias called **info**:

```
annette@thebeach
gidget@hawaiian
frankie@avalon
yourself@yourhost
```

To establish this list, the entry at the end of `/usr/lib/aliases` appears as follows:

```
info: annette@thebeach, gidget@hawaiian, frankie@avalon, yourself@yourhost
```

4. When you have added all user aliases, write the changes to the file, and return to the `#` prompt.
5. At the `#` prompt, enter `newaliases` to rebuild the random access data base for the `/usr/lib/aliases` file.
6. At the `#` prompt, enter `exit` to exit root status and return to normal user status.

Note: Step 5 must be performed each time the distribution list is modified.

As the size of the `/usr/lib/aliases` file increases, `newaliases` takes longer to rebuild the random access data base. Therefore, use the procedure described in "Creating an External Distribution List Alias," below, for creating larger distribution lists.

Creating an External Distribution List Alias

This type of alias distribution list is suitable for establishing a list of many aliases (such as an entire department) or that change frequently.

To create a distribution list alias that refers to an external file, perform the following steps.

1. At the prompt, open a new file to contain the distribution list, using your favorite editor (such as `vi`).
2. Enter each user's alias to be included in the distribution list, using one line per alias.

3. When you have added all user aliases, write the changes to the file, and return to the prompt.
4. At the prompt, enter `su` to change to root status and display the `#` prompt.
5. At the `#` prompt, open the `/usr/lib/aliases` file to edit it.
6. Open a new line at the bottom of the file, then add the list alias name and the pathname of the external file containing the user aliases, using the following format:

```
<list_alias_name>: :include: /<pathname>/<filename>
```

Example: To establish an alias called **grouptalk** for users included in the file **grouptalk.listing** in `/usr/myhost/janice/listings`, the line appears as follows:

```
grouptalk: :include:/usr/myhost/janice/listings/grouptalk.listing
```

7. Write the changes to the file, and return to the `#` prompt.
8. At the `#` prompt, enter `newaliases` to rebuild the random access data base for the `/usr/lib/aliases` file.
9. At the `#` prompt, enter `exit` to exit root status and return to normal user status.

Note: Step 8 does not have to be run each time the external distribution list is modified. This is a major advantage for users who maintain large distribution lists that change frequently.

Delivery of messages to aliases included in an external distribution list is slightly longer than to aliases included in a distribution list contained within the `/usr/lib/aliases` file.

Creating an Alias to Receive Mail

To create an alias so you can receive mail by different names, perform the following steps.

1. At the prompt, enter `su` to change to root status and display the `#` prompt.
2. At the `#` prompt, use your favorite editor to open the `/usr/lib/aliases` file.
3. Open a new line at the bottom of the file, then add the following text.
 - a) The name you wish to receive mail by, followed by your regular alias
 - b) The `owner-` specification, followed by your regular alias
(This line directs error messages resulting from the receive mail alias to your regular alias address)

Example: To establish `gene` as an alias for `gene@themovies`, the `/usr/lib/aliases` entries appear as shown below.

```
gene:      gene@themovies
owner-gene: gene@themovies
```

4. Write the changes and return to the `#` prompt.
5. At the `#` prompt, enter `newaliases` to rebuild the random access data base for the `/usr/lib/aliases` file.
6. At the `#` prompt, enter `exit` to exit root status and return to normal user status.

Note: The `owner-` specification can also be used to direct any alias inquiries to the person responsible for maintaining the list.

Additional Information

Additional information can be found under *aliases(5)* and *newaliases(8)* in the *UNIX Interface Reference Manual*, part number 800-1303.

Cross Compilers

Sun Cross-Compilers 2.0 Announcement

Introduction

This article is a brief overview of Sun Cross-Compilers release 2.0.

Cross-Compilers provide developers with the ability to produce executable binaries for Sun-2, Sun-3, and Sun-4 series system architectures from a single machine. Thus, Cross-Compilers release 2.0 supports cross-development between **host** systems (the systems used for application development) and **target** systems (the systems for which code is being developed), as shown below.

HOST SYSTEM	TARGET SYSTEM
Sun-2 (68010-based)	Sun-3 (68020-based)
Sun-2 (68010-based)	Sun-4 (SPARC-based)
Sun-3 (68020-based)	Sun-2 (68010-based)
Sun-3 (68020-based)	Sun-4 (SPARC-based)
Sun-4 (SPARC-based)	Sun-2 (68010-based)
Sun-4 (SPARC-based)	Sun-3 (68020-based)

The cross-compilation tools are complete in that they consist of all compiler passes, libraries, and include files needed for each combination of host and target architectures.

Applications

There are two major applications for Cross-Compilers, as follows.

First, in situations in which access to a target system of a specific architecture is limited, Cross-Compilers allow binaries for the target system to be conveniently produced on another system architecture. The binaries can then be tested on the target system when it becomes available. For example, Cross-Compilers were used by third-party software developers porting their software to the new Sun-4 architecture prior to field availability of Sun-4 systems.

Second, the productivity of the software developer can be increased by using Cross-Compilers on high-performance 'compute servers' on the network, instead of using native compilation tools on local workstations. For example, diskless Sun-3 systems would see a significant decrease in compilation time by using a Sun-4 on a network for Sun-3 binary production, using Sun-4 to Sun-3 Cross-Compilers.

Supported Languages/Release Levels

Supported languages and release levels include Fortran (F77), Pascal (pC), and C (cC), as shown in the table below.

HOST	TARGET - 1	TARGET - 2
Sun-2	Sun-3: f77 from SunOS 3.4 pc from SunOS 3.2L cc from SunOS 3.4	Sun-4 f77 from SunOS 3.2L pc from SunOS 3.2L cc from SunOS 3.2L
Sun-3	Sun-2 f77 from SunOS 3.4 pc from SunOS 3.2L cc from SunOS 3.4	Sun-4 f77 from SunOS 3.2L pc from SunOS 3.2L cc from SunOS 3.2L
Sun-4	Sun-2 f77 from SunOS 3.4 pc from SunOS 3.2L cc from SunOS 3.4	Sun-3 f77 from SunOS 3.4 pc from SunOS 3.2L cc from SunOS 3.4

NOTE: The Pascal compiler is compiled from SunOS 3.4-compatible code from the Motorola 680X0 version of SunOS 3.2L (the beta version of Sys4-3.2).

Usage

Once Cross-Compilers are installed, usage is simple. An additional option, *-target* (*-sun2*, *-sun3*, or *-sun4*), becomes available on the host system for the following commands:

- cc
- pc
- f77

For example, the following is an example of a Sun-3 native compilation command.

```
%cc a.b b.c -o c.out
%
```

This command on a Sun-3 host becomes a cross-compilation for a Sun-4 target by adding the *-sun4* option, as follows.

```
%cc -sun4 a.c b.c -o c.out
%
```

As another example, the following command on a Sun-4 host can be used to produce Sun-3 binaries, using Sun-4-to-Sun-3 Cross-Compilers.

```
%make CC="cc -sun3"
%
```

Executable Image Compatibility When running Cross-Compilers on a host machine under a specific SunOS version, note that the executable images that are produced are only compatible with that same SunOS version on the target machine. For example, Cross-Compilers running on a Sun-3 host under SunOS 3.x will produce only Sun-3 Release 3.x binaries, and Sun-4 Release 3.x binaries.

Disk Space Requirements Cross-Compilers release 2.0 requires the following space to support each host and target, as shown below.

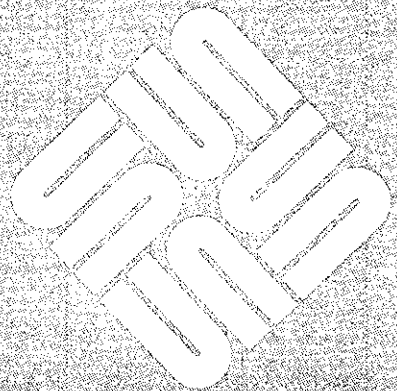
Configuration	Hosts Supported	Targets Supported	Host/Target Combinations	Disk Space Required
Cross-Compiler Server	1	2	2	25 MB
Cross-Compiler Server	3	3	6	37 MB
Cross-Compiler Client	1	2	2	12 KB

Server configuration installations of this product may require more space than is available on a standalone system with a single small (71 MB) SCSI disk.

To ensure maximum disk space efficiency, users may want to plan their use of Cross-Compilers release 2.0 to share common files among multiple languages, host architectures, and target architectures (the three host/target server installations shown above).

STB SHORT SUBJECTS

STB SHORT SUBJECTS	219
Setting MAXUSERS	219
<i>Domain</i> Definitions	221





STB SHORT SUBJECTS

Setting MAXUSERS

The MAXUSERS Setting

This short subject contains details on the kernel table sizes that result, in part, from the MAXUSERS setting you specify during the kernel configuration portion of installing SunOS on your workstation. This information is applicable for SunOS releases 2.x and 3.x.

This setting is key to allowing your kernel to have properly-sized internal tables. Before kernel configuration, MAXUSERS is set to 4. It is a good idea to change this setting to 8 or more when using a bitmap or `suntools`. See the article 'Hints and Tips #9' appearing on page 1024 of the December 1987 STB issue, part number 812-8701-11, for more information.

Another good rule of thumb is to set MAXUSERS to the average number of windows you usually run in `suntools`.

`/usr/sys/conf/GENERIC`

The MAXUSERS setting is tunable in your customized kernel configuration file you made from your copy of the default file `/usr/sys/conf/GENERIC`. This setting is then used to calculate UNIX internal tables described in the following paragraph. Note that MAXUSERS does *not* specify the maximum number of users who can `login` to the machine.

`/usr/sys/conf/param.c`

The equations shown below appear in the file `/usr/sys/conf/param.c` and use MAXUSERS as one factor to set up the UNIX internal tables.

Note that modifications to this file are no longer in force when you run `config`. Any changes you make to this file must be redone whenever you modify the kernel configuration file.

It is possible to modify the equations in this file directly, if desired. However, this level of tuning is rarely necessary. Moreover, it can result in problems since your modifications are hidden in this file rather than in the more visible kernel configuration file.

Thus, we do not recommend that you modify this file, and if you do, we strongly recommend that you document what you have done.

$$\square \quad nproc = (10 + 16 * MAXUSERS)$$

This sets the size of the process table to determine the maximum number of processes you can run at one time.

$$\square \quad ntext = 24 + MAXUSERS$$

This sets the size of the text table to determine how many different binaries you can run at one time.

$$\square \quad ninode = (NPROC + 16 + MAXUSERS) + 64$$

This sets the maximum number of inodes. Note that MAXUSERS is used twice in computing ninode since it is also used to calculate NPROC.

$$\square \quad nfile = 16 * (NPROC + 16 + MAXUSERS) / 5 + 64$$

This sets the maximum number of open files you can use on a system not using the SunView window system.

Or,

$$nfile = 16 * (NPROC + 16 + MAXUSERS) / 10 + 64$$

for systems using the SunView window system.

$$\square \quad ndquot = (MAXUSERS * NMOUNT) / 4 + NPROC$$

This sets the maximum number of quotas. Again, note that MAXUSERS enters this equation twice since it is also used in setting NPROC.

Summary

System administrators should keep the importance of the MAXUSERS setting in mind when configuring the customized kernel. Use of the standard SunOS configuration file `/usr/sys/conf/GENERIC` without increasing MAXUSERS may cause problems, especially when using `suntools`.

Domain Definitions

Domain: Multiple Definitions

The term *domain* is used in several contexts and may have a different meaning in each. This short subject details the three most frequently used definitions of the term.

Please note that this short subject describes the use of the term *domain* per se, and does not attempt to describe details of the examples used. Also, note that the examples themselves overlap each other, even when the term *domain* is used in different contexts.

Addressing Domain

An addressing domain is a group of users or processes that agree on a particular address format. That address format must be used within its domain, to the exclusion of all others. Further, users and processes cannot freely mix addressing formats across domains, even when the addressing domains partially overlap.

Three common examples illustrate this point. The phone company, the postal service, and the United Parcel Service (UPS, in the United States) are examples of addressing domains. Telephone numbers (including area codes) are within the addressing domain of the telephone company. Street addresses (including ZIP codes) and post office box numbers are contained within the addressing domain of the postal service. Finally, street addresses (but *not* post office box numbers) are found within the UPS address domain.

Addressing domains may partially overlap as in the case of the postal service and UPS use of street addresses, but *not* the use of post office box numbers. Another partial overlap is the case of street addresses appearing in the phone book. However, addresses cannot be intermixed nonetheless. You may use the area code when calling, but must not use the street address number, even if it appears in the phone directory.

Domain-Based Addressing

Here the term *domain* describes an area or context in which a particular address is used. Such areas or contexts may be arranged in a vertical hierarchy. The requirement is that within each area, context, or layer within a hierarchy, the names or addresses are unique and not duplicated within that area.

Unique addresses are then distributed within each domain or subdomain within a greater domain. This layering of domains results in a large address that can be broken into pieces or subcodes, each subcode being unique within its subdomain.

Telephone numbers serve as the best example of domain-based addressing. Each area code must be unique. However, two different area codes may each contain identical exchange numbers. Finally, within a particular exchange, each line number must be unique. However, the same line number may be used in any other exchange of the same or any other area code.

Address-Family *Domain*

In network addressing, the address-family domain is defined by the socket location. You specify the address-family associated with that socket. The address-family name is then passed down to lower layers. This allows networking requests to be passed to the appropriate servers, and the like.

For example, educational institutions and commercial corporations each have a different address-family domain, *.edu* and *.com*, respectively. Then within the commercial domain, each corporation has another unique address-family domain. This is typically the name of the corporation, as in *sun.com* for one example.

Finally, a particular network user or alias has a unique name within the company. *stb-editor!sun.com* indicates three layers of address-family domains, the STB editor within Sun Microsystems within the commercial domain.

IN DEPTH

IN DEPTH	225
Graphics Standards	225



IN DEPTH

Graphics Standards

Graphics Standards

Porting graphics applications has often been a difficult problem for companies supporting more than one graphics computer. Each vendor's computer usually runs a proprietary graphics interface that is not only unique to the vendor, but often times unique to the particular machine.

Sun Microsystems has helped to ease the difficulty of porting graphics code by offering numerous graphics standards for Sun workstations. By programming to the standards, a Sun customer is guaranteed compatibility across all Sun products, as well as with other vendor's machines which support the standard.

In order to ensure portability across different graphics machines, a graphics standard establishes one agreed-upon set of graphics commands. These commands provide a common interface to different graphics machines. Thus, programmers can use the standards knowing that their graphics program will run on a variety of machines.

The History Of Graphics Standards

The history of high-level graphics standards is the history of two rival families of standards, Core and GKS, each trying to become *the* graphics standard. See figure 1 for a summary of the history of graphics standards.

The seeds for Core were sewn in 1974 at a workshop held by the National Bureau of Standards (NBS). From this workshop emerged the Graphics Standards Planning Committee (GSPC), which operated within the Association of Computing Machinery's Special Interest Group on Graphics (ACM/Siggraph). In 1977 the GSPC presented the Core system, a device-independent graphics application interface, to the graphics industry. Two years later, using Core as a starting point, the American National Standards Institute (ANSI) established a committee for computer graphics programming languages, X3H3, to work on a graphics standard.

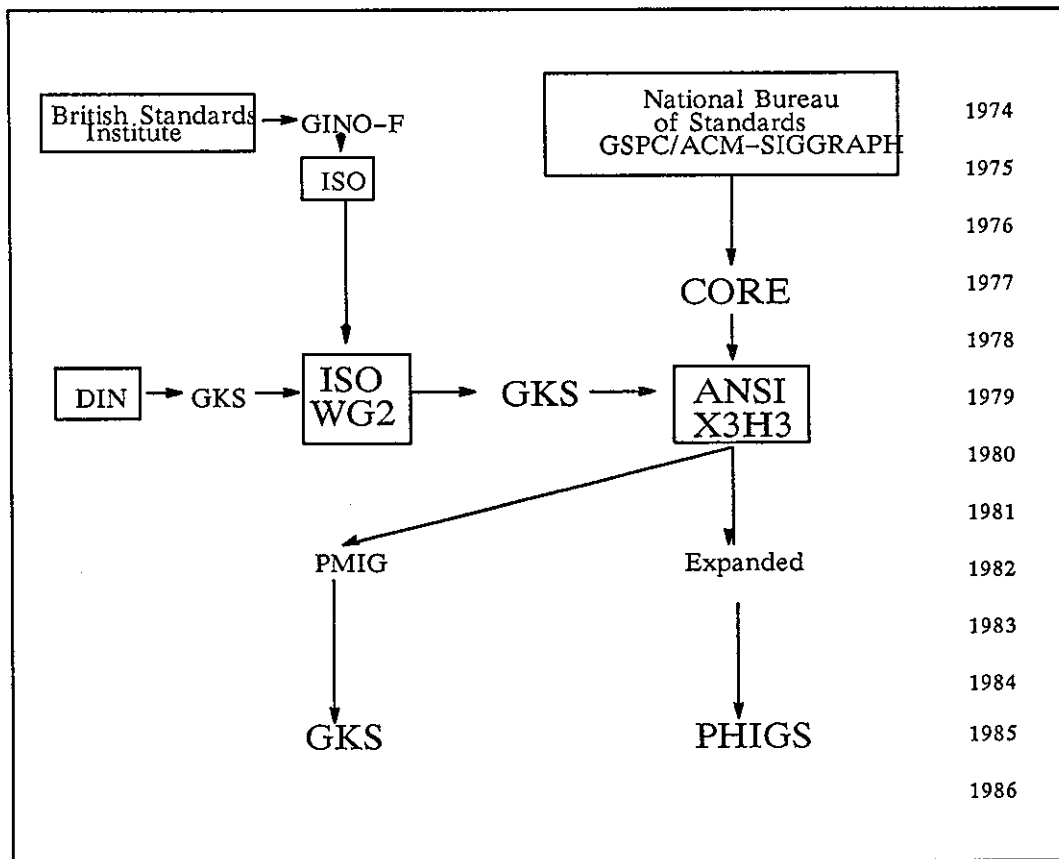


Figure 1: Graphics Standards History 1974-1986

Meanwhile, the efforts which would eventually result in GKS were under way. In 1975, the International Standards Organization (ISO) accepted the British Standards Institute's proposal on graphics standards, GINO-F. While GINO-F itself is not important, it prompted the ISO to create a working group, WG2, to develop a graphics standard. In 1979, two years after Core was proposed, the Deutches Institut fur Normung, the West German standards body, proposed the Graphics Kernel System (GKS) to WG2 for consideration as a graphics standard. Since GKS was submitted after Core, it included many of the strengths of Core while eliminating many of its weaknesses.

In an effort to unify the standards, GKS was presented in 1979 to the the ANSI committee, X3H3, which was using Core as its model for a graphics standard. X3H3 turned GKS down, preferring instead to expand Core. By 1980, X3H2 had split into two camps. One, represented by the personal computer manufacturers, wanted a small, simple graphics standard. This group devised a standard known as the Programmer's Minimal Interface to Graphics (PMIG).

The other group, represented by manufacturers with heavier graphics demands, preferred a more encompassing graphics standard. This group's efforts went into a standard known as the Programmer's Hierarchical Interactive Graphics Standard (PHIGS). Finally, in 1982, X3H3 decided that GKS, which had become popular in Europe, should serve as the group's standard. Additionally, the committee merged PMIG and GKS, making PMIG a subset of GKS. Thus, by 1985, there were three widely-accepted, high-level graphics standards: Core, GKS, and PHIGS.

While Core, GKS, and PHIGS provide standard graphics interface with a high level of functionality, there is also a need for a standard interface at a low level, much closer to the graphics hardware. With such a low-level interface, one standard set of device drivers can be written which would support a wide range of graphics devices.

In 1980, ANSI began a standards effort to establish VDI and VDM standards. It formed the Task Group X3H33. Work on the two standards proceeded in parallel, but in May 1982, X3H33 suspended the VDI effort to permit the VDM specification completion. One reason for pushing forward on the metafile was that it was easier to standardize. It is easier to understand, and it achieves a common transport mechanism among diverse graphics devices. More important, it makes no judgement on how the device interprets the data it reads from the metafile.

In 1982, ANSI submitted its VSM standards effort to ISO TC97 SC5 which in turn formed the Working Group 2 Metafiles Subgroup. By the end of 1985, the VDM and VDI had become both ISO and ANSI standards. In the process, their names had been changed. VDM had become Computer Graphics Metafile (CGM) and VDIO had become the Computer Graphics Interface (CGI). CGI is the answer to the many computer-graphics peripherals manufacturers who want one common interface that will enable their equipment to quickly attach to a computer system.

SunCGI, SunGKS, SunCore, and FIGARO

Since each type of application requires its own level of graphics functionality, numerous graphics standards have emerged, ranging from a simple device interface to a complex three-dimensional imaging model.

At the lowest level, the Computer Graphics Interface (CGI) serves as a general interface to specific graphics devices. Moving up a step from the hardware, the Graphics Kernel System (GKS) provides the ability to group two-dimensional graphics primitives together into segments, route graphics output to different display surfaces, and read and write graphics to files of a standard format. The

ACM/Siggraph Core provides a slightly higher level of functionality by adding 3D as well as 2D graphics.

And finally, PHIGS has the most sophisticated standard imaging model to date. On top of the two- and three-dimensional graphics primitives, PHIGS offers a hierarchical display list to aid in managing graphics data. Sun supports CGI, GKS, and Core directly with the SunCGI, SunGKS, and SunCore packages; a PHIGS package honed to Sun's hardware, FIGARO, is available at present from Template.³

See figure 2 for an illustration of the Sun implementation of the CGI, GKS, Core, and PHIGS imaging models.

Performance and Windows

Since performance is critical to the success of standards, and windows are vital to the ease of graphics applications use, Sun has tightly coupled the standards to its hardware accelerators and has fully integrated the standards into the Sun windowing environment. Sun has implemented the standards so that there is a minimal path leading from the standard's software interface to Sun's graphics hardware.

To ensure convenient integration of graphics applications into Sun's windowing environment, Sun has augmented the standards to embrace a multi-tasking and multiple-window graphics display. Within Sun standards, integration into a windowing environment is transparent to the applications.

SunCGI

SunCGI provides a two-dimensional graphics interface to low-level graphics routines. SunCGI has neither the restrictions nor the overhead of higher level standards like GKS or Core; nor does it have many of their features. SunCGI delivers the efficiency and freedom of a device driver to the application programmer at the expense of high level functions.

The most notable functions absent from CGI are explicit transformations, three-dimensional functions, and segmentation. If these features are required by an application, the programmer must use either another standard or write the procedures.

Imaging Model

The imaging model for standard CGI is based upon the concept 'virtual device'. The virtual device is conceptually a generic graphics device to which standard CGI serves as an interface. This virtual device serves as a common model for devices ranging from pen plotters to CRTs. Standard CGI provides the programmer with the input and output primitives needed to control the virtual device.

SunCGI extends the standard CGI concept of the virtual device by allowing one program to open numerous view surfaces on one device. Since SunCGI is integrated with Sun's windowing environment, this extension allows an

³ Template is a registered trademark of Megatek Corporation, 9645 Scranton Road, San Diego, CA 92121

application to draw to a number of windows which act as view surfaces. These view surfaces can be manipulated as normal Sun windows; the application or a user can position them, change their size, and close them into icons.

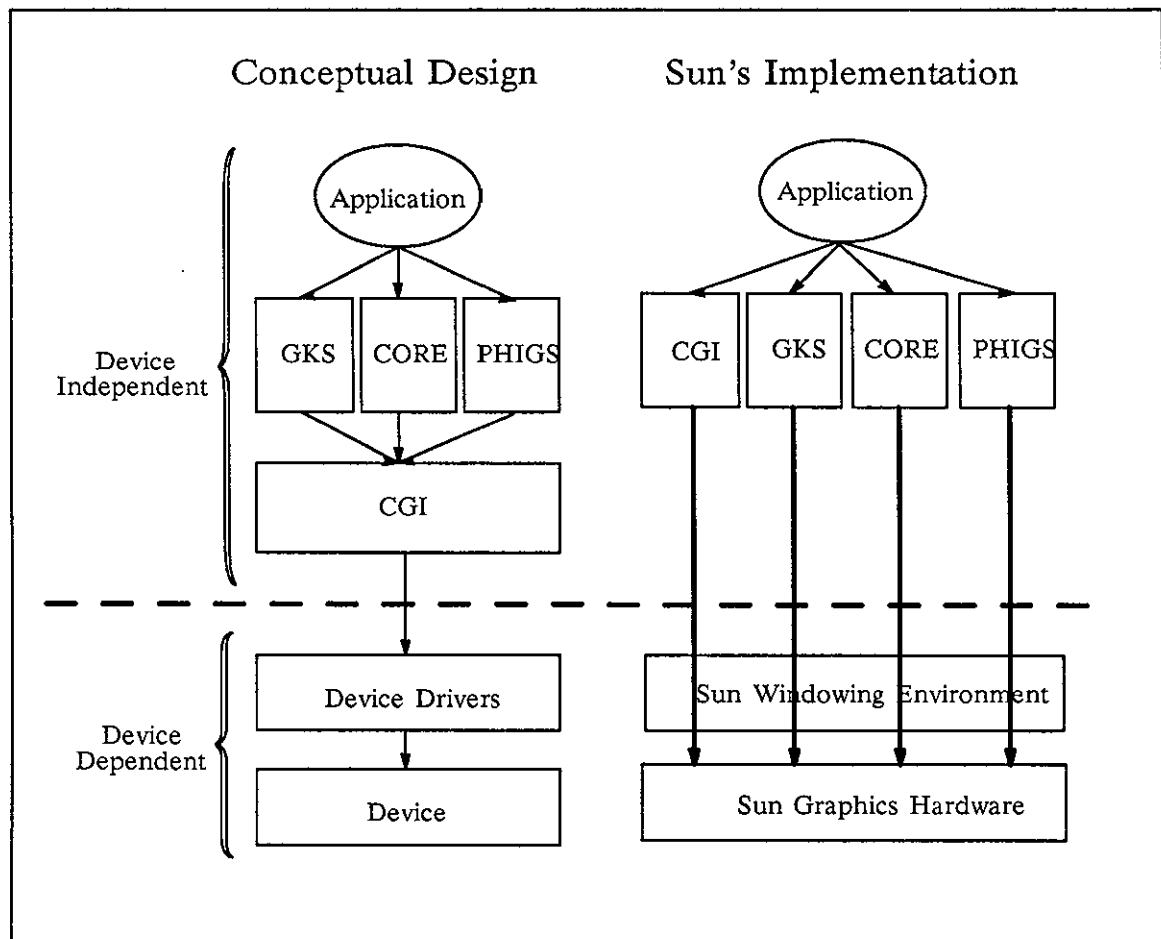


Figure 2: Sun Implementation of Graphics Models

View Surfaces

SunCGI allows a single process to open, close, activate, and de-activate multiple view surfaces as shown in figure 3. Opening a view surface initializes the surface to a default state. Activating a view surface ensures that subsequent SunCGI calls will affect that surface; nothing can be drawn on a surface that has not been activated.

When a view surface is created, it has its own color map (except when the view surface is on a monochrome display) and name.

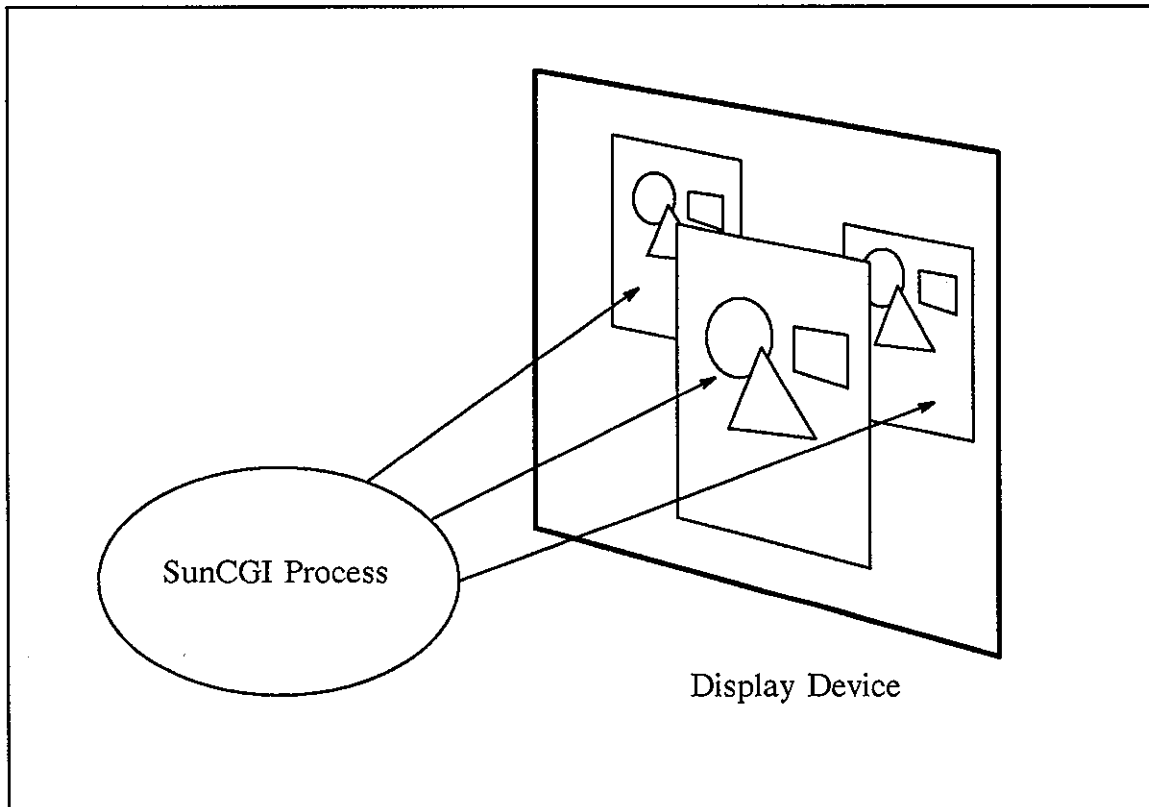


Figure 3: SunCGI Processes and Multiple View Surfaces

Input Model

An input device consists of a trigger and a measure. A trigger is a device used to mark a point in time. For example, a trigger is the button on the mouse. A measure is the current value of the input device. The measure of a mouse would be the x and y coordinate cursor locations on the screen. When a device is triggered, the device causes an input event. An input event is the combination of a measure from an input device, coupled with the identification of the input device. In the mouse example, the input event consists of the x and y mouse locations, and the identification of the mouse button that triggered the event.

There are five classes of input devices: locators, strokes, valuator, choices, and strings. A locator picks an (x,y) point in Virtual Device Coordinate (VDC)

space; a stroke returns an array of (x,y) points in VDC space; a valuator returns a normalized x position; a choice returns a number representing a selection from a number of choices; and, finally, a string returns a character string. SunCGI uses the mouse as a locator, stroke, valuator, and choice input device. The keyboard serves as a string input device.

SunCGI supports synchronous and asynchronous input. For synchronous input, the input device is set to the `RESPOND_EVENT` mode. In this mode, when an application requests a measure from an input device, the application waits until the input device returns a value. To get asynchronous input, the input device can be set to either `REQUEST_EVENT` or `QUEUE_EVENT`.

When the input device is set to `REQUEST_EVENT`, the application can initiate asynchronous communication between it and the process handling the input. Each time a new event is created by the input device, it is placed in the *request register* where the event can be polled by the application. If a device is set to `QUEUE_EVENT`, each new event associated with the input device is placed into an *event queue*.

The event queue is a First In First Out (FIFO) buffer shared by all input devices. As input events are created they are placed on the queue where the application can search the buffer for the last event or events from a specific input device.

Coordinate System

The coordinate system the application uses in SunCGI is called Virtual Device Coordinates (VDC). In conventional computer graphics terms, VDC space corresponds to World Coordinate (WC) space. These integer coordinates can be set by the application and can range from -32762 to 32767. Primitives drawn in VDC are mapped to the physical devices coordinates by SunCGI. See figure 4 for an illustration of SunCGI mapping from VDC to screen space.

The mapping from VDC space to the physical screen space is isotropic; the aspect ratio of the image in VDC determines the aspect ratio of the image on the physical device.

By manipulating the extent of the virtual device coordinates, an application can translate (move) or scale (zoom in or zoom out) an image. No rotation functions are provided by SunCGI, so they must be supplied by the application program.

Clipping and Transformations

There are no user-defined transformations in SunCGI. Zooming and panning can be achieved by manipulating the VDC extent. The clipping bounds can be turned off or defined by the application. Turning off the clipping speeds up drawing graphics primitives at the risk of drawing them outside the view surface window. An arbitrary clipping rectangle can be set in VDC by the application.

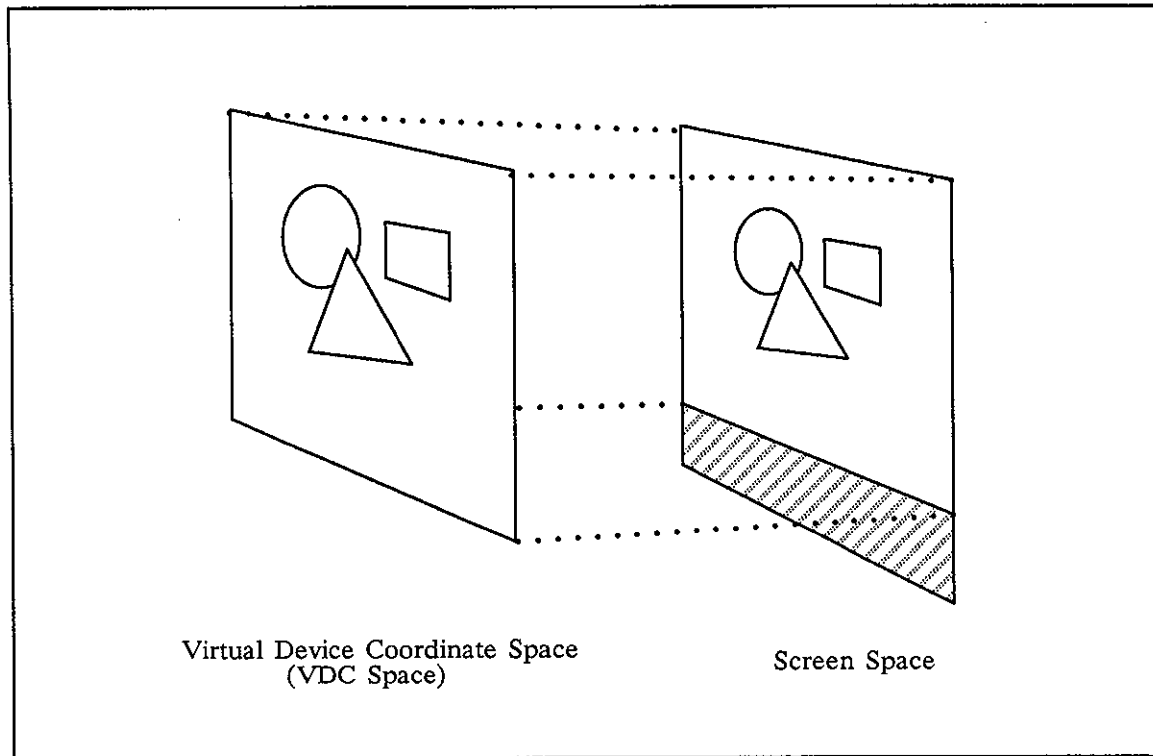


Figure 4: SunCGI Mapping from Virtual Device Coordinate to Screen Space

Raster Functions

Raster primitives include text, cell arrays, pixel arrays, and bit block transfers (bitblts). The text primitives allow the application to choose a font, scale the characters, set the spacing between letters, and set the orientation of the text. A cell array draws a scaled and skewed pixel array on the screen; a cell array can be drawn into a parallelogram. A pixel array draws a rectangle of pixels onto the screen. Finally, bitblts copy an array of pixels from one area on the screen to another. In addition to merely writing raster primitives, it is also possible to repeat a raster pattern to fill an area on the screen. Since many raster objects are dependent on the screen resolution, a number of raster operations must be defined in screen space, not VDC.

In SunCGI, it is possible to perform logical operations on raster primitives as they are drawn to the screen by combining the source pixels and the destination pixels with logical operators. SunCGI allows rasters to be considered 'transparent' as they are written; to be `anded`, `ored`, `notted`, `xored`, or drawn normally as they are written to the screen (i.e., to be `notted` before they are combined on the screen).

These logical raster operations give the application several capabilities used to manipulate text and areas of pixels quickly and effectively. SunCGI extends this concept beyond the standard CGI by allowing all graphics primitives to be written to the screen with logical operations.

Geometric Primitives

In addition to raster primitives, SunCGI includes many geometric primitives: arcs, circles, ellipses, polylines, rectangles, polygons, markers, polymarkers, and text. Each is described in the following paragraphs.

□ Arcs

There are two broad types of arcs, circular and elliptical. For both types, SunCGI allows the application to create closed arcs, that is arcs that define an area. A closed arc can be filled in either `PIE` or `CHORD` mode as shown in figure 5. In `PIE` mode, lines are drawn from the end-points of the arc to the center of the circle or ellipse. The result looks like a piece of pie or a section of a pie chart. In `CHORD` mode, the end-points of the arc are joined by a line. The resulting regions are filled according to the current solid object attributes.

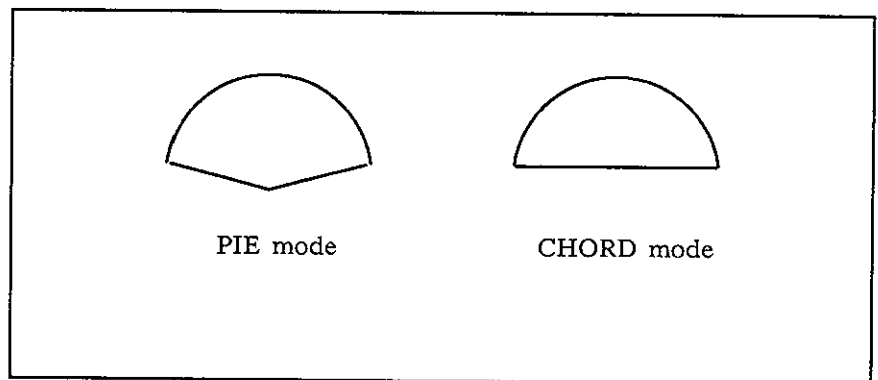


Figure 5: The PIE and CHORD Methods Used to Close Arcs

Circular arcs can be defined by two methods. One is by specifying the center of the circle, its radius, and the two arc end-points. The other

method defines a circle by specifying three points through which the circle must pass.

Open arcs are drawn according to the current line attributes.

□ *Circles*

Circles are drawn by specifying a point and a radius. The interior and perimeter of the circle are controlled by the current solid object attributes.

□ *Ellipses*

Ellipses are drawn by specifying a center and the major and minor axes. The interior and perimeter of the ellipse is controlled by the current solid object attributes.

□ *Polylines*

The polyline function draws a list of connected lines. The last point is *not* automatically connected to first point. To generate a closed polyline, the last point on the list must have the same coordinates as the first. The disjoint polyline function draws a list of disjoint polygons; both end-points of each line need to be specified. The line appearance is controlled by the current line attributes.

□ *Rectangles*

A special function is available to draw a rectangle by defining its lower, right corner and its upper, left corner. The interior and perimeter of a rectangle are defined by the current solid object attributes.

□ *Polygons*

A polygon can be drawn by specifying a list of points. Polygons can be self-intersecting and can have holes in them. The interior and perimeter of the polygon are controlled by the current solid object attributes.

□ *Text*

SunCGI provides a full set of text functions. Text can be generated in one of three modes. If the 'text precision' is set to be `STRING`, the firmware character set is used. In this case, the characters can be neither rotated nor scaled. In this mode, individual characters are not clipped; either a whole character is written within the clipping bounds or it is not written at all.

If the text precision is set to `CHARACTER`, characters are generated in software. In this mode, clipping is similar to the clipping in `STRING` mode. And finally, when the text precision is set to `STROKE`, the

CHARACTER text precision attributes are used and individual characters can be clipped in parts.

SunCGI offers a variety of fonts including Roman, Greek, Script, Olde English, stick, and symbols. In addition, SunCGI allows the application to set the spacing between characters, the character height, and the text color. The text skew and direction can be determined as well as the character path (right, left, up, or down). To give further flexibility, SunCGI allows the application to determine the point within the character that is used to define the character's position.

Sun has extended the CGI text facilities by adding a feature to set characters to be either fixed- or variable-width. If the characters are fixed, all the characters are of uniform size; if the characters are variable, the characters are spaced according to their respective sizes.

- *Polymarkers*

Polymarker draws a tick mark at a list of points. The marker can be a dot, a plus sign, an asterisk, a circle, or an 'X'. Additionally, the marker's size and color can be set via the polymarker attributes.

Attributes

The following paragraphs describe SunCGI attributes including line, solid object, and bundled attributes.

- *Line Attributes*

Line attributes are a set of global variables which affect the appearance of polylines and arcs. Once they are set, they define the appearance of subsequent polylines and arcs. They do not affect the outlines of solid objects since the perimeter attributes perform this function.

The line attributes and options are summarized in figure 6. The available line attributes are line type (solid, dotted, dashed, dashed-dotted, dash-dot-dotted, and long dashed), line end-style, line width, and line color. Line end-style is a Sun extension to CGI which allows the application to determine how gracefully to process the end-points of a textured (non-solid) line.

- *Solid Object Attributes*

Solid object attributes determine how circles, closed arcs, ellipses, rectangles, and polygons are filled. There are three general classes of solid object attributes: fill attributes which determine how the object should be filled, hatch and pattern attributes which determine the type of pattern that is used for a pattern fill, and perimeter attributes which determine the appearance of the boundary of the solid object.

Solid objects can be hollow (not filled), or filled with a solid color, a pattern, or a hatch. A pattern is different than a hatch in that it can be

scaled and translated; a hatch simply fills a solid object with an arrays of pixels.

<u>Attribute Type</u>	<u>Options</u>
Line Attributes	Line Type Line Endstyle Line Width Line Color
Perimeter Attributes	Perimeter Type Perimeter Width Perimeter Color
Polymarker Attributes	Marker Type Marker Size Marker Color

Figure 6: SunCGI Line Attributes and Options

If an object is solid and not hollow, its interior color can be set. If an object is filled with a pattern, the pattern size in VDC can be set. SunCGI also allows patterns and hatches to be stored in a table so that the current pattern or hatch can be changed by altering an index.

Sun has extended the pattern and hatch attributes to allow a pattern to be applied transparently over an object filled with a solid color. Using this feature, it is possible to define 'holes', or transparent areas, where the object's underlying solid color is visible.

Finally, the perimeter of the object can be defined using the perimeter attributes. The available perimeter attributes are perimeter type (solid, dotted, dashed, dashed-dotted, dash-dot-dotted, and long dashed), perimeter width, and perimeter color.

□ *Bundled Attributes*

A group of output primitives frequently share the same set of attributes. Each time one of these primitives is drawn, each individual attribute has to be set. To make this process more efficient, SunCGI allows the application to bundle attributes into one package. This way, when a set of attributes must be set, they can be accessed through one primitive.

Colors

In SunCGI, colors are defined by integer indices into a color table. This color table defines the set of colors that can be seen on the screen at the same time. SunCGI allows the application to define the color table.

SunGKS

The Graphics Kernel System (GKS) is a device independent, two-dimensional graphics interface which lies between an application and a device driver. See figure 7. Since GKS serves as an interface to an application, its focus is to make writing applications more convenient. GKS incorporates abstractions which are common to a wide variety of graphics applications, but specific to none. Here GKS differs from the Computer Graphics Interface (CGI). While GKS is looking upward to the applications for its model, CGI looks downward towards the device for its model. GKS is thus more convenient to use for applications.

Because of its focus on applications, GKS offers features which are not present in CGI such as metafiles (standard input and output files), segments, transformations, floating point coordinates, and well-developed control over multiple workspaces. GKS is a convenient interface for writing two-dimensional, interactive graphics applications. The price for this convenience is that SunGKS programs are larger and run less quickly than SunCGI programs.

If an application requires three-dimensional graphics primitives, SunGKS is not the standard to use; both SunCore and PHIGS process three-dimensional graphics. For two-dimensional applications, however, SunGKS provides a convenient and portable 2c graphics interface, providing GKS functionality for both input and output. The 2c graphics interface input and output levels are shown in figure 8.

Imaging Model

The imaging model behind GKS is broken into two distinct sections: one to create graphical objects and specify their characteristics, and the other to determine how these objects will be rendered on a specific physical device. Another way of viewing this distinction is that one part of the imaging model provides graphics application programmers with a convenient interface, the other part ensuring that an application will run on a variety of devices.

To aid the application in creating and manipulating graphical objects, GKS provides an imaging model based on *segments*. Segments are groups of graphical primitives (lines, polygons, and so forth) and their attributes which create a coherent object. Segments allow an application programmer to manipulate a complex graphical object without worrying about its constituent primitives. Since segments are central to the imaging model, much GKS programming reduces to creating, manipulating, describing, and deleting segments.

Since graphics devices vary widely, from plotters to graphics terminals to graphics workstations, GKS uses a *workstation* to serve as a generic input and output device.⁴ By defining a generic workstation, GKS shields the application

⁴ A *workstation* in GKS is one graphics process. A workstation corresponds to one window or device on a Sun computer.

from the peculiarities of specific devices. All of the idiosyncrasies of a particular device are introduced in the translation from the workstation to the physical device.

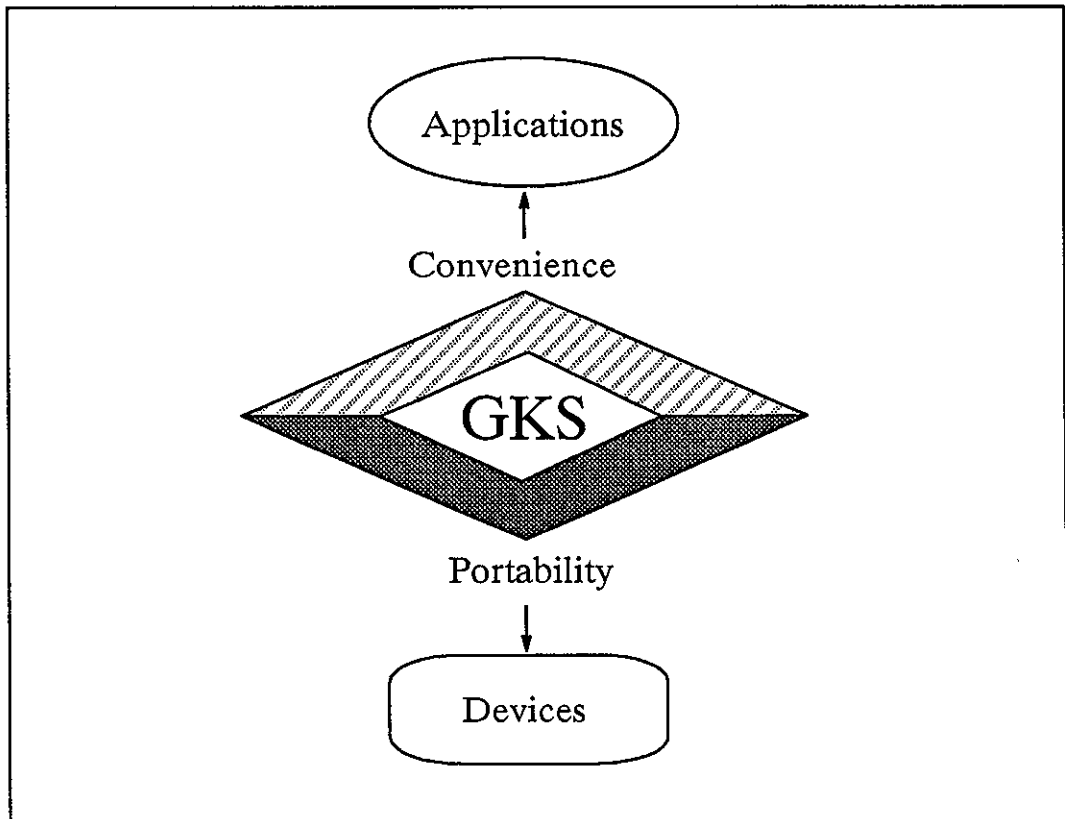


Figure 7: The SunGKS Graphics Interface

Worksurfaces

Since SunGKS is closely integrated with the Sun windowing environment, managing workstations is closely related to managing Sun windows. With this relationship in mind, there are five different types of workstations that a GKS program can write to: Workstation Independent Segment Storage (WISS), Metafile Input (MI), Metafile Output (MO) ASCII format, SunView 'window', and a SunView 'canvas'. GKS allows you to write efficiently to different windows. You can set the size and locate the GKS workstation on the screen.

Input Model

SunGKS uses the same input model as CGI and includes more functions for inquiring the state of items relating to interface negotiation and input capabilities. Refer to sections 2.4 and 2.5 of the *SunCGI Reference Manual*, part number 800-1256, for details.

Coordinate System

The application defines the World Coordinates (WC). A transformation to a Normalized Device Coordinate (NDC) window is implicitly defined by setting the window in WC and the viewport in NDC. See figure 9 for an illustration of SunGKS transformations from WC to NDC to screen coordinates.

Input Level	Output Level
a) No input functions allowed	0 Minimal input; one settable normalization; one workstation
b) Only Request input allowed	1 More than one output workstation, setments, and multiple normalizations
c) Request, Sample, and Event allowed	2 Workstation independent storage allowed

Figure 8: The SunGKS 2c Support Level

The SunGKS workstation transformation is affected both by standard GKS functions, and implicitly by operator changes to the window size. The workstation transformation maps an area in the NDC space within the range $[0.0,1.0] \times [0.0,1.0]$ (the workstation window) onto a specified area of the device coordinate space (the workstation viewport). Segments are stored in NDC space.

The device coordinates used to address the window are fixed at $[0,4095] \times [0,4095]$ and are independent of the size or shape of the window, and are independent of the physical device on which the window appears.

Clipping and Transformations

Clipping can be turned on and off. There are two sets of clipping. The first set is to the workstation window and cannot be turned off. The second set is to the normalization transformation viewport. This second set can be turned on and off.

The programmer can also attach a rotation, scaling, and translation matrix to a segment. This has the effect of moving, scaling, or rotating the image generated by the segment. SunGKS maintains an array of normalization transformations. The application program can select and recompute these normalization

transformations. The current normalization transformation is used for all workstations. This allows the programmer to switch between workstations by changing the normalization matrix.

The segment transformation function uses a 2 x 3 transformation matrix that determines the rotation, scaling, and translation of a segment. GKS provides two utility functions for generating a transformation matrix from various components and for generating the Accumulate Transformation Matrix for altering an existing transformation.

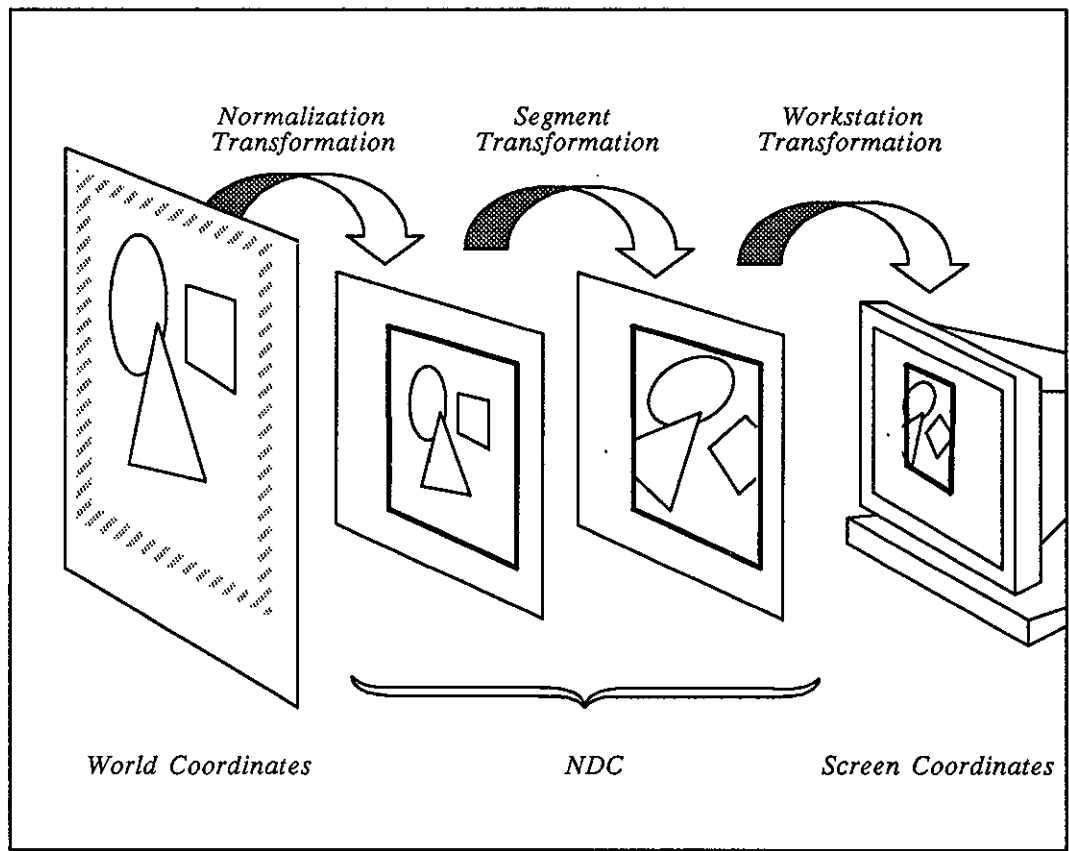


Figure 9: SunGKS Transformations from World to Screen Coordinates

The parameters used in transformations are listed below.

<i>fixed point</i>	indicates the origin for the segment coordinate system
<i>shift</i>	how much to shift in the <i>x</i> and <i>y</i> directions
<i>angle</i>	angle of rotation for the segment coordinate system
<i>scale</i>	how much the segment coordinate system is expanded or contracted
<i>coordinates</i>	whether the fixed point and shift vector parameters are in WC space or NDC

Segments

A segment is a static data structure for storing groups of output primitives and attributes. A GKS application describes a graphical object by creating a segment, calling output primitive functions and attribute functions (the results of which are placed in the segment), and then by closing the segment. Segments have names (integer identifiers) so that a GKS application can selectively modify parts of a complex model by deleting and recreating segments (which effectively replaces them).

SunGKS uses the current attribute list at segment creation time. Attribute functions included in the segment have their normal effect. However, once the segment is closed, the primitive attributes used in its creation cannot be changed.

Segments have five segment attributes as listed below.

- transformation
- visibility
- highlighting
- priority
- detectability

These attributes allow the programmer to recreate, delete segment from a workstation, insert segment, set segment priority, and the like.

Raster Functions

Raster functions allow the programmer to inquire pixel values on the screen, either singly or as an array of values. Also, cell array facilities are available.

Metafile

If a GKS workstation is of type Metafile Output (MO), all objects written to it are stored in a special GKSM format disk file. A Metafile Input (MI) GKS workstation can read this metafile. Metafiles can be copied around the network, over internetwork routers to other networks, accessed by other GKS graphics

application software, or used as a standard storage format for drawings. The metafile is in clear text, an ASCII file.

Hardcopy support is added to GKS by a special workstation whose connection is output-only. An alternative method is to provide a program that reads a metafile and generates hardcopy. To obtain a hardcopy, use the `screendump` program or the `pixrect` rasterfile I/O routines.

Geometric Primitives

Geometric primitives include polylines, text, and polymarkers. These are shown below along with their attributes.

□ *polylines*

line width
 line type
 polyline index
 linetype
 linewidth scale factor
 polyline color index
 linetype aspect source flag
 linewidth scale factor aspect source flag
 polyline color index aspect source flag

□ *text*

alignment
 color index
 font and precision
 index
 path
 representation
 expansion factor
 height
 spacing
 up vector

□ *polymarkers*

color
 index
 representation

Color Color is represented by an index into a color table. This color table can be set by the application.

Fill Areas are filled explicitly by a `fill` command. The fill type is determined by the current fill attribute. The programmer can set the items shown below.

color
 interior style (hollow, solid, pattern, hatch)
 pattern

pattern size
 pattern reference point (starting point of the pattern)

SunCore

Core is the grandfather of graphics standards. Developed in the mid-1970s by an ACM/Siggraph committee, Core was the first attempt at an industry-wide graphics standard. Since Core was developed as computer graphics was coming of age, much of Core is aimed at the functionality of three-dimensional vector displays. In 1979, however, Core was extended to include raster functions. Sun has extended Core even further to include z-buffer hidden surface removal, lighting models, and Gouraud and Phong shading.

Core differs in scope from CGI and GKS in that it focuses on three-dimensional applications. As a result, much of the Core standard centers around specifying a three-dimensional view. Core is similar to GKS in that it provides segments to group geometric primitives. As is the case with GKS, Core conveniently uses floating point coordinates.

SunCore supports the ACM Core output level 3c (dynamic output including two-dimensional (2D) and three-dimensional (3D) translation, scaling, and rotation). Additionally, SunCore supports the ACM Core input level 2 (synchronous input, including the PICK logical device). And, finally, SunCore supports dimension level 2 (3D operations). The SunCore output levels, input levels, and dimension levels supported are shown in figure 10.

Imaging Model

Since SunCore is a 3D graphics standard, most of its imaging model centers on defining the position of graphical objects, the camera view from which the objects are seen, and finally how the view will be projected into two dimensions.

The three-dimensional position of an object and the camera view from which it is seen are determined by two different types of matrices. The *modeling transformations* determine an object's position in three dimensions. Since each object has a unique position, Core assigns a modeling transformation to each graphical object. The *viewing transformation* determines a viewpoint and the characteristics of the camera.

As shown in figure 11, a number of parameters must be defined in order to specify a view. The first parameter is the *eye point*, or sometimes known as the *view point* or *center of projection*. In conjunction with the eye point, a *viewing plane* must be specified. The viewing plane is the plane onto which the 3D graphical objects will be projected.

<u>Output Level</u>	<u>Input Level</u>	<u>Dimension Level</u>
1) Basic Output	1) No Input	1) 2D
2) Buffered Output	2) Synchronous Input	2) 3D
3) Dynamic Output	3) Complete Input	
3 a) 2D translation		
3 b) 2D Scale, rotation, and translation		
3 c) 3D scale, rotation, and translation		

Figure 10: SunCore Output, Input, and Dimensions Supported

The viewing plane is conceptually similar to the plane in a camera where the film rests. The viewing plane is defined by the *view plane normal*, which is the vector normal to the viewing plane which passes through the *view reference point*. Since the viewing plane has infinite boundaries, the next step is to define a rectangular clipping boundary on the view plane. This boundary, the *window*, defines the field of view.

The term 'window' in SunCore is not the same as the term 'window' when it is used in the context of a windowing environment. The orientation of the window is set by the *view up vector*. The window has a local coordinate system with the axes *u*, *v*, and the view plane normal. The corners of the window are specified in *uv* coordinates.

The *front clipping plane* and the *back clipping plane* are defined by the *front distance* and the *back distance*. The front clipping plane is sometimes called the 'hither' plane, and the back clipping plane is sometimes called the 'yon' plane. Objects closer to the eye point than the front clipping plane and objects farther from the eye point than the back clipping plane are not shown in the final picture.

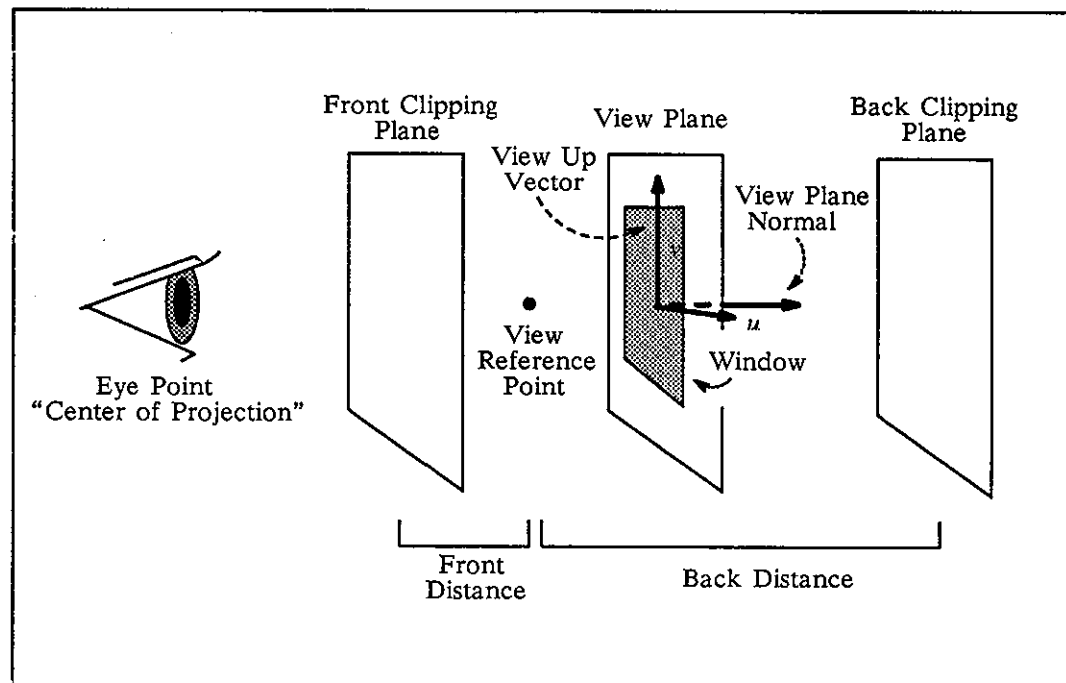


Figure 11: SunCore View Points

And finally, the last parameter need to specify fully the view is the *projection flag*. SunCore supports perspective projection and parallel projection. SunCore provides defaults for each of these viewing parameters, but all of them can be set by the application.

After the view has been set, the transformation from the window to the device must be defined. The area on the device where the final picture is shown is called the *viewport*. The viewport is specified in Normalized Device Coordinates (NDC). Since SunCore is well-integrated into the Sun windowing environment, a SunCore device corresponds to one Sun window in the windowing system. The viewport is a region within a Sun window. The final viewing transform is created by packing all of viewing parameters, along with the viewport specification, into one matrix. See figure 12 for an illustration of mapping the viewing window to the viewport within a Sun window.

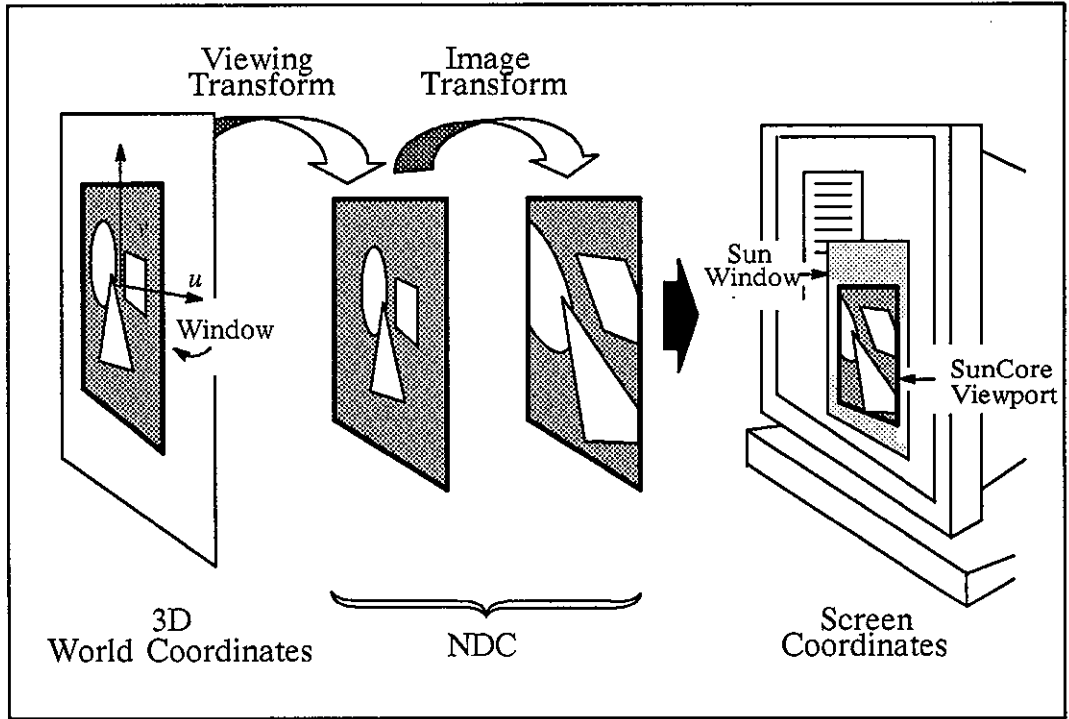


Figure 12: SunCore Mapping of View Windows to Viewports

In summary, the SunCore imaging model is similar to taking a picture with a camera. First, the objects are put into place by the modeling transformations. The properties of the camera are selected. Then, with the objects in place and the camera defined and pointed, the viewing transformations project the image of the 3D objects onto the 2D viewing plane. The image in the viewing plane is then shown on the display device.

Worksurfaces

Since SunCore is a 3D package, the concept 'workspace' is removed from the surface of the 2D display device. SunCore's workspace is the window within the viewing plane.

Coordinate Systems

Application programs which draw pictures using SunCore communicate in World Coordinates (WCs). World coordinates are a device-independent, 2D or 3D, Cartesian coordinate system for describing objects. Output primitives are given to SunCore routines in WCs. However, if the world coordinate matrix is used, SunCore concatenates this matrix with the view transform so that output

primitives are first transformed by this matrix from 'model' or 'object' coordinates to world coordinates. This means that the user can supply primitives in 'model' coordinates, each model or object being moved into world coordinates according to the current world coordinate matrix.

In 3D, the user may choose to use right-handed or left-handed world coordinates.

The composite viewing transform is formed from the world coordinate matrix and the viewing parameters. SunCore routines transform the output primitives from world (or model) coordinates to NDC, which is a left-handed coordinate system bounded such that: $0.0 < x, y, z < 1.0$.

Since current Sun view surfaces have three-to-four aspect ratios, the default NDC space has the y extent bounded to $0.0 < y < .75$. Primitives are stored in a display list in NDC space. The user-specified window in world coordinates is mapped, and optionally clipped, to the user-specified viewport within NDC space. The entire NDC space is then mapped to the selected physical view surfaces.

Input Model

The input model uses synchronous input. That is, when the program expects input, it halts until either it times out or input is received. Echoing facilities are provided that allow you to attach a particular echo to a particular input device.

Refer to the input capabilities table in the 'Introduction', chapter 1 in the *SunCore Reference Manual*, part number 800-1257, for further information.

Segments

ACM/Core provides 3D graphics package capabilities. SunCore is level 3c (dynamic output with 3D scaling, rotation, and translation) for output primitives, and is level 2 (complete synchronous input) for input primitives. The extensions to Core are textured polygon fill algorithms, raster primitives, RasterOp attributes, shaded surface polygon rendering, and hidden surface elimination.

Segments are a group of primitives that are treated as a whole. Typically, a segment might correspond to one specific part of a greater whole, like a tire of a car. In SunCore, there are two types of segments: retained segments and temporary segments. Retained segments are named and are kept throughout the duration of the Core application. Temporary segments are only drawn once; they cannot be modified dynamically.

When a retained segment is created, it can be one of five types. The most flexible is the transformable, 3D type of segments. This segment contains 3D data that can scaled, rotated, and translated. The next type of segment is a 3D translatable segment. This type can be only translated in space, neither rotated nor scaled. Next is the 2D transformable segment. This type of segment can be scaled, rotated, and translated in two dimensions. The 2D translatable segment can be only translated in 2D. Finally, there is the fixed segment which cannot be transformed.

Every retained segment has four dynamic attributes: visibility, highlighting, detectability, and an image transformation. Visibility determines whether the

image should be visible or not. Highlighting specifies whether the segment's image should be highlighted. In SunCore, highlighting is done by blinking the object. Detectability indicates whether a picking input device can detect the segment. And finally, the image transformation indicates how the image of a retained segment is scaled rotated or translated.

You can manipulate retained segments as listed below.

- open segment (only one segment can be opened at a time)
- close segment
- name
- rename
- delete
- save on disk
- restore from disk

Raster Functions

You can put a raster which draws a rectangular 1 bit- or 8 bits-deep raster, and enters it into the current segment. A raster primitive may, however, be picked or dragged if it is entered in a translatable segments. You can either store a 1 bit- or an 8 bit-per-pixel raster. Also, you can read a raster area from the screen. The raster-to-file capability allows you to copy a raster file to disk.

You can also read a raster from a raster file and can inquire how large a raster is, given the NDC coordinates and the output device. RasterOps `xor` and `or` are available.

Geometric Primitives

The following geometric primitives are available.

- *lines*

- linestyle
- linewidth
- absolute and relative positioning

- *polygons*

- polygon edge style
- interior style
- absolute and relative positioning

- *text*

- character size
- spacing (additional space between strings)
- chara up (slanting, mirror imaging of characters), 2D and 3D
- charpath (which direction the characters go), 2D and 3D
- character precision.

- *current position*

- move in absolute coordinates
 - move in relative coordinates

- *markers*

- marker symbol
 - marker in absolute and relative coordinates in both 2D and 3D
 - polymarkers in absolute and relative coordinates in both 2D and 3D

Colors

A user-settable color table is used for colors. You can set the color to be used to fill areas, lines, and can set the color used for text and markers. You do this as a global function, not when the function is called.

Special 3D Operations

z-buffer hidden surface, flat, Gouraud, and Phong style shading is supported. The Phong lighting model is implemented, as shown in the equation below.

$$\text{pixelshade} = \text{ambitnet} + \text{diffuse}(L*N) + \text{specular}(H*N)**\text{bump} - (\text{flood}*z)$$

You can set the direction of the light source. Also, you can set the vertex normals for Phong shading, and the colors in Gouraud shading.

The Programmers Hierarchical Interactive Graphics Standard

The Programmers Hierarchical Interactive Graphics Standard (PHIGS), has the broadest functionality of any of the current industry standards. In addition to providing a common interface for describing and drawing geometric primitives, PHIGS provides a standard for managing a hierarchical graphical database. Core, CGI, and GKS all revolve around drawing something on the screen. PHIGS standardizes the database. PHIGS defines a graphics support system for the creation, modification and display of graphical objects. In PHIGS, structures relate to each other in a hierarchical network. Structures can reference, or execute, other structures.

PHIGS main contribution is the standardization of graphics data. Hierarchy, modeling and editing capabilities are supported. Specifically it manages the storage and display of 2D and 3D graphical data. PHIGS creates and maintains a hierarchical database, relieving the application program of a great deal of unnecessary overhead. PHIGS defines a graphics support system for the creation, modification, and display of graphical objects. An object is defined by a sequence of elements, including output primitives, attributes, and transformations. These elements are grouped into one or more entities called structures.

A structure consists of a set of structure elements. Each element represents a unique unit of graphical information. Output primitives, attributes, modeling transformations, view selections, pick identifier, labels, name sets, structure invocations, application data, and escape functions are supported. Non-retained structures are not maintained in a database since they are rendered directly. Retained structures are stored for further modification by PHIGS.

PHIGS allows the dynamic modification of graphical images. Specifically, an application can edit structure contents, perform database operations on a structure, and modify attribute values in workstation state lists. These powerful operations give applications control over the generation of possible real-time display images and flexibility over the management of database structures and their contents.

You open the structure to edit it and close the structure after editing. You can insert elements, delete elements, and copy structures. You can also traverse through the structures which are stored in a tree-type arrangement. Finally, you can empty a structure by deleting all of its elements, delete it outright, or rename it.

Structure coordinates use modeling coordinates, the same as object space, which are transformed to world coordinate space coordinates. The structure inherits the attributes of its ancestors unless another value is defined within the current structure.

Primitives include the polyline, polymarker, text, fill area, cell array, and generalized drawing primitive. Input includes the locator, stroke, valuator, choice, pick, and string. Operating modes include request and sample event. Finally, picking is supported which allows the application to know which part of the hierarchy has been selected. PHIGS returns the whole picking path to distinguish different instances of the same object.

PHIGS and GKS Differences

Structures and the centralized, hierarchical graphics database are processed differently. An important and powerful capability of PHIGs, lacking in GKS is the ability to edit structures by inserting and deleting structure elements. This allows simple changes to a graphic image as well as complex changes to its hierarchical description.

GKS requires the application program to supply hierarchical modeling capabilities and limits dynamic operations to groups of primitives stored after the view operation. GKS groups primitives into entities called segments which reside in a decentralized, mono-level database. Data descriptions are not hierarchical; segments cannot invoke other segments. GKS segments can be neither extended nor edited.

PHIGS supports traversal-time binding, and permits inheritance and dynamic modification of primitive attributes through structure editing. The application program can therefore change the enhancements without changing the data structures. In GKS, primitives are assigned attributes at definition time. Enhancements cannot be applied without changing the data structures.

Summary

See figure 13 on the next page for a summary of SunCGI, SunGKS, and SunCore comparisons.

	SunCGI	SunGKS	SunCore
Input	Synchronous Asynchronous	Synchronous Asynchronous	Synchronous
Dimension of Data	2D	2D	2D, 3D
Coordinates	VDI [-32767,32767] Integer	World Coordinates, NDC, DC Floating Point	World Coordinates, NDC, DC Floating Point
Clipping	Define Clipping Rectangle	Define Clipping Rectangle	Define Clipping Rectangle
Transformations	none	2D Transformations	2D, 3D Transformations
Raster Functions	BitBlt, scalable cell array		RasterOps, raster to file get raster
Segments	no	yes	yes
Geometric Primitives	circle ellipse arcs lines polylines polygons text markers polymarkers _	- - - lines polylines polygons text markers polymarkers -	- - - lines polylines polygons text markers polymarkers current position

Figure 13: SunCGI, SunGKS, and SunCore Comparison Summary



QUESTIONS, ANSWERS, HINTS, AND TIPS

QUESTIONS, ANSWERS, HINTS, AND TIPS	255
Q&A, and Tip of the Month	255

Copyright © 2000
All Rights Reserved



QUESTIONS, ANSWERS, HINTS, AND TIPS

Q&A, and Tip of the Month

Hints & Tips #10

This is the tenth in a continuing series of this column which I have created for two purposes.⁵ First, some questions are asked regularly on the AnswerLine. I feel everyone can benefit from distributing discussions of these problems as widely as possible. Second, a large and constantly growing body of information, hints, and tips are not documented anywhere.

I will collect and distribute these information nuggets in this continuing column so that we can all learn from them. I will cover unusual topics, but this column should not be used as an alternative to contacting your support center or using the AnswerLine.

If you have a question that you would like answered in this column, please mail your question to 'Software Technical Bulletins' at Sun Microsystems, Inc., 2550 Garcia Avenue, M/S 2-312, Mountain View, CA 94043. You can also send in your question by electronic mail to *sun/stb-editor*. U. S. customers can call Sun Customer Software Services AnswerLine at 800 USA-4-SUN for technical questions on this column or any other article in this bulletin. I look forward to hearing from you!

UUCP and Your USERFILE

When setting up a UUCP connection, the most common place for errors seems to be the USERFILE. The USERFILE is the protection system for UUCP that sets up the local and remote site restrictions in copying files in and out of the system.

The most frequent problem seen with the USERFILE is that the administrator forgets to modify it for their system. As it comes off the distribution tape, the USERFILE looks appears as shown below.

⁵ This continuing column is submitted by Chuq Von Rospach, Customer Software Services.

```
,sun /
, /usr/spool/uucppublic
```

This allows the system named *sun* to copy anywhere in the directory tree, but all other sites can only copy in and out of the publicly accessible directory `/usr/spool/uucppublic`. The first line needs to be changed so that the hostname *sun* is changed to the local machine hostname. If you do not do this, you will run into permission problems and the system will not work properly.

uucp Bug in SunOS Releases 3.x

Another common problem with the `USERFILE` occurs when the administrator tries to relax the permission restrictions. There is a bug in `uucp` versions 3.x where it turns off all permissions to everyone if there is only a single line in `USERFILE`.

You would expect that the line shown below would allow anyone to copy anything. In fact, it will not allow anything to happen. Avoid the line shown below.

```
, /
```

The Workaround

The workaround to this is simple, fortunately; just duplicate the line as shown below.

```
, /
, /
```

For the real security conscious, be aware that there is an upper limit to the number of lines you can put in the `USERFILE`. The internal structures that are used to process the `USERFILE` can support only 20 lines in the file. If you do exceed this limit, `UUCP` will start failing with a `'BAD USERFILE'` error message.

Tip of the Month

A very common request to Customer Service Division is how to set up the mail system to share a common `/usr/spool/mail` file via NFS. If you have several machines where a user is likely to use any of the machines (as in a laboratory setting, for example), it would be nice if they could get their mail without having to `rlogin` to a specific host. Currently, SunOS releases 3.x do not support this feature, but it is easy to implement if you are careful.

The key to making a central NFS-mounted `/usr/spool/mail` file work is to make sure that `sendmail` never goes over the wire, since it is `setuid` to root. Instead, you want to make `sendmail` use the SMTP delivery protocols to transfer the mail to the `sendmail` process on the machine that has the local disk, thereby avoiding the root-over-NFS problem.

To do this, you need to make sure that all mail is sent to the host with `/usr/spool/mail` on the local disk. This can be done using the mail aliases file to cause all mail to be forwarded to the central host. For every user in the

`/etc/passwd` file, add an alias into the file `/usr/lib/aliases` as shown below.

```
user: user@mailserver
```

In this example, *mailserver* is the name of the machine with the local disk. For simplicity and performance, it is usually a good idea to make this machine the mailhost.

Once this is set up, each system can use NFS to mount the central mail directory, using the command shown below.

```
% mount mailserver:/usr/spool/mail /private/usr/spool/mail
%
```

Or, in the `/etc/fstab` file, you can add the line shown below.

```
mailserver:/usr/spool/mail /private/usr/spool/mail nfs rw,soft
```

Once this is done, users should be able to read and delete mail from the central repository.

Other Notes and Caveats

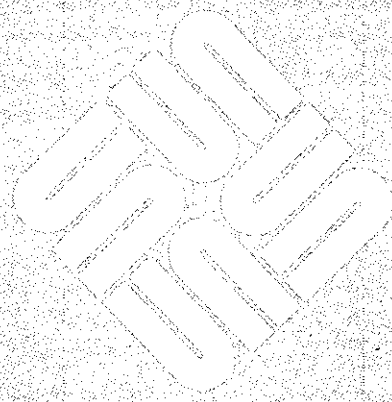
A few notes and caveats on this. First, the `biff` command and the `comsat` daemon do not work with an NFS-mounted mail directory, so users will not get notified when mail arrives. Setting the `mail` variable in the `csh` should work, and can be used as an alternative.

Also, if you start seeing mailboxes in `/usr/spool/mail` owned by 'nobody', there is a problem with the aliases. Either some machine is ignoring them and writing directly over the network, or an alias is missing or set up incorrectly.



THE HACKERS' CORNER

THE HACKERS' CORNER	261
Backup Copy Daemon	261





THE HACKERS' CORNER

Backup Copy Daemon

An Automatic Backup Copy Daemon

This month's **Hackers' Corner** contains a shell archive for an Automatic Backup Copy Daemon (ABCD).

Please consult your local shell script or programming expert regarding any script or code problems. The example programs are not offered as a supported Sun product, but as items of interest to enthusiasts wanting to try out something for themselves. Note that **Hackers' Corner** code may not work in all cases, and may not be compatible with future SunOS releases.

The Automatic Backup Copy Daemon

This Automatic Backup Copy Daemon (ABCD) continually copies modified files from one filestore area to another using either `cp` or `rccp`. Note the second filestore area may be on another machine if the file system is NFS-mounted.

The backup area should therefore have an identical copy of the filestore being monitored.

The ABCD README

See the README text contained in first and second pages of the archive for details on the following subjects.

- Backup area filestores
- Notes on ABCD processing
- Available ABCD switches

The ABCD Archive

The ABCD archive appears on the following pages, including details on ABCD usage appearing in the README file.

```

#!/bin/sh
#
# This is a shell archive.
#
# 1. Save this archive text in a file.
# 2. Execute the file with '/bin/sh' to create the following files:
#
#  README
#  Makefile
#  abcd.c
#  abcd.h
#
export PATH; PATH=/bin:$PATH
#
if [ -f README ]
then
echo shar: will not over-write existing file README
else
echo shar: extracting 'README',      2333 characters
cat > README <<'Funky_Stuff'

```

```

ABCD
----
```

This is an Automatic Backup Copy Daemon, which copies files from one filestore area to another using either 'cp' or 'rcp'. Note the second filestore area may be on another machine if the file system is NFS-mounted.

The backup machine should have an identical copy of the filestore being monitored.

NOTES

- (1) A start directory for monitoring is given. All of the files including sub-directories and their files are monitored. When any file is changed, ABCD automatically copies it to another machine as specified by the initial parameters to the program.
- (2) If ABCD finds a new directory, it will make that new directory in the backup area on the other machine if not already present.
- (3) If you delete a sub-directory, then ABCD removes that directory from the list of monitored directories. If you remove the original start directory, then the program terminates since there is nothing left to monitor.
- (4) If you have made no changes during one complete pass of all monitored directories, then ABCD sleeps for 5 minutes.
- (5) There is a restart facility whereby if the machine has crashed, it is possible to tell ABCD not to copy anything for the first pass,

but to start copying changed files from the second and subsequent passes.

- (6) ABCD uses the 'system' system call to do its copying, so as to prevent spawning of child processes.

SWITCHES

-
- fdirectory Directory to start copying from. Defaults to '/usr'.
 - tdirectory Directory to start copying to. Defaults to '/usr2'.
 - rhostname This is an alternative form of backup. It uses 'rcp' and copies to the supplied hostname.
 - sseconds Specifies the sleep period in seconds when ABCD puts itself to sleep.
 - c Causes ABCD to not copy during the first pass.

```
Funky_Stuff
len=`wc -c < README`
if [ $len != 2333 ] ; then
echo error: README was $len bytes long, should have been 2333
fi
fi # end of overwriting check
if [ -f Makefile ]
then
echo shar: will not over-write existing file Makefile
else
echo shar: extracting 'Makefile', 842 characters
cat > Makefile <<'Funky_Stuff'
#
# Makefile for ABCD, an Automatic Backup Copy Daemon.
#
#
```

```
BINARIES      = abcd
BINDIR        = .
CFLAGS        = -g
LDFLAGS       = -g
OBJJS         = abcd.o
SRCS          = abcd.c
HDRS          = abcd.h
LIBS          =

all:          $(BINARIES)

release:      $(BINARIES)
              strip $(BINARIES)
              mv $(BINARIES) $(BINDIR)
```

```

backup:
    cp abcd.c abcd.c
    cp abcd.h abcd.h

clean:
    rm -f abcd *.o *.c core

lint:
    lint $(SRCS) $(LIBS)

abcd:
    $(OBJS)
    cc $(LDFLAGS) -o abcd $(OBJS) $(LIBS)

abcd.o:
    abcd.c $(HDRS)
Funky_Stuff
len=`wc -c < Makefile`
if [ $len !=      842 ] ; then
echo error: Makefile was $len bytes long, should have been      842
fi
fi # end of overwriting check
if [ -f abcd.c ]
then
echo shar: will not over-write existing file abcd.c
else
echo shar: extracting 'abcd.c',      12888 characters
cat > abcd.c <<'Funky_Stuff'

/* abcd.c
 *
 * This is an Automatic Backup Copy Daemon, which copies files from one
 * filestore area to another using either 'cp' or 'rcp'. Note the second
 * filestore area may be on another machine if the file system is NFS-mounted.
 *
 * The backup disk should have an identical copy of the filestore being
 * monitored.
 *
 * SWITCHES
 *
 * -fdirectory Directory to start copying from. Defaults to '/usr'.
 *
 * -tdirectory Directory to start copying to. Defaults to '/usr2'.
 *
 * -rhostname This is an alternative form of backup. It uses 'rcp'
 * and copies to the supplied hostname.
 *
 * -sseconds Specifies the sleep period in seconds when ABCD puts
 * itself to sleep.
 *
 * -c If given, causes ABCD to not copy during the first pass.
 */

```



```

#include <stdio.h>
#include <strings.h>
#include <errno.h>
#include <signal.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "abcd.h"

char *malloc(), *sprintf() ;

struct finfo *cfile ; /* Information on current file. */
struct finfo *files = NULL ; /* Files being monitored. */
struct dinfo *cdir ; /* Pointer to current directory. */
struct dinfo *dirs = NULL ; /* Directories being monitored. */
struct stat cstat ; /* For statistics on current file. */
struct direct *cur ; /* Pointer to current directory record. */

extern int errno ; /* Standard error reply. */

char cname[MAXLINE] ; /* Copy program to use. */
char coff[MAXLINE] ; /* Name offset from initial start directory. */
char curdir[MAXLINE] ; /* Current directory being monitored. */
char dbuf[DIRBLKSIZ] ; /* Buffer for directory information. */
char fdir[MAXLINE] ; /* Directory to start copying from. */
char flist[MAXFILES][MAXLINE] ; /* Array of filenames to copy. */
char fname[MAXLINE] ; /* Full pathname of current file. */
char pdir[MAXLINE] ; /* Current directory for copy request. */
char rhost[MAXLINE] ; /* Name of remote host for use by rcp. */
char tdir[MAXLINE] ; /* Directory to start copying to. */

int copy_found = 0 ; /* Whether copy done on this pass. */
int delay = 300 ; /* Sleep time for abcd in seconds. */
int docopy = 1 ; /* 'rcp' files right from the beginning. */
int fd = 0 ; /* File id of the directory being monitored. */
int loc = 0 ; /* Current location in the directory block. */
int fcnt = 0 ; /* Number of files to copy in current request. */
int no_dirs = 1 ; /* Number of directories being monitored. */
int size ; /* Size of current directory block in bytes. */
int usercp = 0 ; /* Backup method, 'cp' or 'rcp'. */

get_options(argc,argv) /* Get ABCD options from command line. */
int argc ;
char *argv[] ;

{
char *arg ;
char *p ; /* Pointer to string following argument flag. */

STRCPY(fdir, "/usr") ; /* Default directory to monitor. */
STRCPY(tdir, "/usr2") ; /* Default directory to copy to. */

```

```

STRCPY(rhost, "");          /* Default is no remote host. */

while (argc > 1 && (arg = argv[1])[0] == '-')
{
    p = arg + 2 ;
    switch (arg[1])
    {
        case 'c' : docopy = 0 ;          /* Do not copy files the first time. */
                    break ;
        case 'f' : STRCPY(fdir,p) ;      /* Directory to start monitoring from. */
                    break ;
        case 'r' : STRCPY(rhost,p) ;     /* Name of remote host. */
                    usercp = 1 ;
                    break ;
        case 's' : delay = atoi(p) ;     /* Sleep period in seconds. */
                    break ;
        case 't' : STRCPY(tdir,p) ;      /* Directory to start copying to. */
                    break ;
        default  : FPRINTF(stderr,"USAGE: abcd [-c] [-ffromdir] [-rhostname]");
                    FPRINTF(stderr," [-ssleep] [-ttodir]\n");
                    exit(1) ;
    }
    argc-- ;
    argv++ ;
}

setup()

{
    dirs = (struct dinfo *) malloc(sizeof(struct dinfo)) ;
    STRCPY(dirs->d_name, "");
    STRCPY(pdir, fdir) ;          /* Start directory for current 'cp' request. */
    STRCPY(coff, "") ;          /* Name offset from start directory. */
    if (usercp) STRCPY(cname, "/usr/ucb/rcp") ; /* Used by the output routine. */
    else STRCPY(cname, "/bin/cp") ;
    dirs->next = dirs ;
    cdir = dirs ;
}

get_next_dir()          /* Get next directory name to monitor. */

{
    struct dinfo *temp ;
    int dirfound ;

    dirfound = 0 ;
    if (fd)
    {
        CLOSE(fd) ;
        if (!strlen(cdir->next->d_name)) /* Complete pass done? */

```

```

    {
        docopy = 1 ;                               /* Always copy after first pass. */
        if (!copy_found)
            sleep((unsigned int) delay) ; /* Nothing happening, go to sleep. */
        else copy_found = 0 ;
    }
    if (strcmp(curdir, cdir->next->d_name) != 0)
        if (fcnt) output("", REGULAR) ;
}
do
{
    STRCPY(curdir, fdir) ;
    if (strlen(cdir->next->d_name))
    {
        STRCAT(curdir, "/") ;
        STRCAT(curdir, cdir->next->d_name) ;
    }
    STRCPY(coff, cdir->next->d_name) ;
    if ((fd = open(curdir, 0)) == -1)
    {
        if (EQUAL(curdir, fdir)) exit(0) ; /* Nothing left to monitor. */
        else
        {
            temp = cdir->next ;
            if (cdir->next == dirs) dirs = cdir ;
            cdir->next = cdir->next->next ;
            free((char *) temp) ; /* Lose this directory record. */
            no_dirs-- ;
        }
    }
    else dirfound = 1 ; /* Directory found. */
}
while (!dirfound) ;
cdir = cdir->next ; /* Point to current directory. */
loc = 0 ; /* Reset directory buffer pointer. */
}

```

```

make_dir_entry() /* If not there already, create a new directory record. */

```

```

{
    int i ;
    char tempdir[MAXLINE] ; /* Temporary directory name. */
    struct dinfo *temp ;

    temp = cdir ;
    for (i = 0; i < no_dirs; i++) /* Is the directory already being monitored? */
    {
        temp = temp->next ;
        STRCPY(tempdir, fdir) ;
        if (strlen(temp->d_name))
        {
            STRCAT(tempdir, "/") ;

```

```

        STRCAT(tempdir,temp->d_name) ;
    }
    if (EQUAL(tempdir,fname)) return(0) ;
}

temp = (struct dinfo *) malloc(sizeof(struct dinfo)) ;
temp->next = dirs->next ;
dirs->next = temp ;
dirs = temp ;

STRCPY(dirs->d_name,"") ;
if (strlen(coff))
{
    STRCAT(dirs->d_name,coff) ;
    STRCAT(dirs->d_name,"/") ;
}
STRCAT(dirs->d_name,cur->d_name) ;
no_dirs++ ;
return(1) ;
}

get_next_entry(fd)      /* Get next directory filename entry. */
int fd ;

{
    int tfd ;           /* Temporary file descriptor for file entry. */

    for (;;)
    {
        if (!loc)
            if ((size = read(fd,dbuf,DIRBLKSIZ)) <= 0) return(0) ;
        if (loc >= size)
        {
            loc = 0 ;
            continue ;
        }
        cur = (struct direct *) (dbuf+loc) ;
        if (cur->d_fileno == 0) return(0) ;
        SPRINTF(fname,"%s/%s",curdir,cur->d_name) ;
        if ((tfd = open(fname,0)) == -1)
        {
            loc += cur->d_reclen ;
            continue ;
        }
        else
        {
            CLOSE(tfd) ;
            cur = (struct direct *) malloc(sizeof(struct direct)) ;
            bcopy(dbuf+loc,(char *) cur,(int) DIRSIZ((struct direct *) (dbuf+loc))) ;
            loc += cur->d_reclen ;
            STAT(fname,&cstat) ;
            return(1) ;
        }
    }
}

```

```

    }
}

no_record()      /* Check is this file is already being monitored. */

{
  if (files == NULL) return(1) ;
  cfile = files ;
  do
    if (cfile->direct->d_fileno == cur->d_fileno) return(0) ;
  while ((cfile = cfile->next) != NULL) ;
  return(1) ;
}

make_file_entry()      /* Make a record for this new file. */

{
  struct finfo *temp ;

  if ((cstat.st_mode & S_IFMT) == REGULAR)
  {
    temp = (struct finfo *) malloc(sizeof(struct finfo)) ;
    temp->next = files ;
    files = temp ;

    files->direct = (struct direct *) malloc(sizeof(struct direct)) ;
    files->direct = cur ;
    files->mtime = cstat.st_mtime ;
  }
}

file_modified()      /* Check if this file has been changed. */

{
  if (cfile->mtime == cstat.st_mtime) return(0) ;
  cfile->mtime = cstat.st_mtime ;
  return(1) ;
}

copy(filetype)      /* Copy file to another directory. */
int filetype ;

{
  char name[MAXLINE] ;

  if (docopy)      /* Are we copying this time around. */
  {
    copy_found = 1 ;

```

```

    STRCPY(name, "");
    if (strlen(coff))
    {
        STRCAT(name, coff) ;
        STRCAT(name, "/" ) ;
    }
    STRCAT(name, cur->d_name) ;
    if (strcmp(curdir, pdir) != 0) /* Current dir. different from 'cp' dir.? */
    {
        if (fcnt) output(name, REGULAR) ;
        STRCPY(pdir, curdir) ;
    }
    if (filetype == DIRECTORY)
    {
        if (fcnt) output(name, REGULAR) ;
        output(name, DIRECTORY) ;
    }
    else if (fcnt >= MAXFILES) /* Room in file list for this filename? */
    {
        output(name, REGULAR) ;
        STRCPY(flist[fcnt++], name) ;
    }
    else STRCPY(flist[fcnt++], name) ; /* Save filename in file list. */
}
}

output(name, filetype)
char name[MAXLINE] ;
int filetype ;

{
char command[MAXLINE*7], rdirname[MAXLINE] ;
int i ;

switch (filetype)
{
case DIRECTORY : if (usercp)
    SPRINTF(command, "%s -r %s/%s %s:%s", cname, fdir, name, rhost, curdir) ;
    else
    {
        STRCPY(rdirname, tdir) ;
        STRCAT(rdirname, "/" ) ;
        STRCAT(rdirname, name) ;
        if ((fd = open(rdirname, 0)) == -1)
            SPRINTF(command, "mkdir %s", rdirname) ;
        else return ;
    }
    break ;
case REGULAR : if (usercp) SPRINTF(rdirname, "%s:%s", rhost, curdir) ;
    else
    {
        STRCPY(rdirname, tdir) ;

```

```

        if (strlen(coff))
        {
            STRCAT(rdirname, "/") ;
            STRCAT(rdirname, coff) ;
        }
    }
    STRCPY(command, cname) ;
    for (i = 0; i < fcnt; i++)
    {
        STRCAT(command, " ") ;
        STRCAT(command, fdir) ;
        STRCAT(command, "/") ;
        STRCAT(command, flist[i]) ;
    }
    STRCAT(command, " ") ;
    STRCAT(command, rdirname) ;
}
if (system(command))
    FPRINTF(stderr, "abcd failed: %s\n", command) ;
else FPRINTF(stderr, "abcd succeeded: %s\n", command) ;
FPRINTF(stderr, "\n\n\n") ;

fcnt = 0 ;
}

main(argc, argv)
int argc ;
char *argv[] ;

{
    get_options(argc, argv) ;           /* Get command line options. */
    setup() ;                          /* Initialize parameters. */
    while (MACHINE_WORKING)            /* Do it until the machine crashes. */
    {
        get_next_dir() ;               /* Is there another directory? */
        while (get_next_entry(fd))     /* Is there another file in dir? */
        {
            if (!DOTS(cur->d_name))     /* Is it the . or .. entry? */
            {
                if (no_record())        /* Is this file already monitored? */
                {
                    if ((cstat.st_mode & S_IFMT) == DIRECTORY) /* Directory? */
                    {
                        if (make_dir_entry()) copy(DIRECTORY) ;
                    }
                    else
                    {
                        make_file_entry() ; /* Make a file entry. */
                        copy(REGULAR) ;    /* Copy it to backup machine. */
                    }
                }
            }
        }
        else if (file_modified()) copy(REGULAR) ; /* File been modified? */
    }
}

```

```

        }
        else free((char *) cur) ;
    }
}
}
Funky_Stuff
len=`wc -c < abcd.c`
if [ $len != 12888 ] ; then
echo error: abcd.c was $len bytes long, should have been 12888
fi
fi # end of overwriting check
if [ -f abcd.h ]
then
echo shar: will not over-write existing file abcd.h
else
echo shar: extracting 'abcd.h', 1651 characters
cat > abcd.h <<'Funky_Stuff'

/* abcd.h
 *
 * Definitions used by ABCD, the Automatic Backup Copy Daemon.
 *
 */

#define CLOSE (void) close /* To satisfy lint. */
#define FPRINTF (void) fprintf
#define SPRINTF (void) sprintf
#define SIGNAL (void) signal
#define STAT (void) stat
#define STRCAT (void) strcat
#define STRCPY (void) strcpy

#define DIRECTORY S_IFDIR /* Type of files being monitored. */
#define REGULAR S_IFREG

#define DIRBLKSIZ 512 /* Block size for directory read. */
#define DOTS(A) (A[0] == '.' && (A[1] == 0 || (A[1] == '.' && A[2] == 0)))
#define EQUAL(a,b) !strcmp(a,b) /* Test for string equality. */
#define MACHINE_WORKING 1 /* Forever and a day .... */
#define MAXFILES 5 /* Max no of files to copy in one go. */
#define MAXLINE MAXNAMLEN+1 /* Maximum length of path names. */

struct dinfo /* Information record for directories being monitored. */
{
    struct dinfo *next ;
    char d_name[MAXLINE] ;
} ;

struct finfo /* Information record for monitored files. */
{
    struct finfo *next ;
    struct direct *direct ;

```



```
        time_t mtime ;
    } ;
Funky_Stuff
len=`wc -c < abcd.h`
if [ $len !=      1651 ] ; then
echo error: abcd.h was $len bytes long, should have been      1651
fi
fi # end of overwriting check
```



CUMULATIVE INDEX: 1988

CUMULATIVE INDEX: 1988 277





CUMULATIVE INDEX: 1988



Index

I

1-800-USA-4-SUN
device driver calls, 51

8

800 USA-4-SUN
use of, 12

A

address
device drivers, 48
address mask, 67
addresses
classes of, 107
Internet, 107
alias
used with history, 78
aliases
creating your own, 209
distribution lists, 210
mail, 155
receiving mail, 212
AnswerLine, 9, 155, 255
device driver calls, 51
architecture
Prism, 151
ARP, 109

B

back-to-back packets, 79
backup copy daemon
Hackers' Corner, 261
bind
port numbers, 75
boot
from PROM monitor, 73
booting
specific kernel, 76
Bridge box, 81
broadcasting
subnets, 107
brochure
Sun Education, 26
buffer
Ethernet, 79
buffers
color frame, 140

buffers, *continued*
frame, 37

bug
reporting, 13
bulletin board
Sun Education, 26

C

canvas
colormaps, 146
carrier sense, 114
checksum
Ethernet, 96
client
sample programs, 130
stream socket, 130
collisions
detection of, 115
color, 139
maps, 140
colormaps, 36
configurations
controllers, 59
disks, 59
Sun-2, 62
Sun-3, 60
CONSULT-HSPEED
high-speed disciplines, 52
CONSULT-PLOCK
lock process text, 52
consulting
device drivers, 51
specials, 51
controller
Ethernet, 79
controllers
combinations with disks, 61, 62
disk configurations, 59
SunOS installation, 63
conversion
color to monochrome, 37
copy
backup daemon, 261
courses
device drivers, 56
Sun Education, 26
cross compilers

cross compilers, *continued*
 2.0 announcement, 214
 applications, 214
 compatibility, 216
 disk requirements, 216
 CSD Consulting
 device drivers, 51
 specials, 51
 Customer Software Services, 9
customer-training@sun.com
 Sun Education, 26

D

daemon
 backup copy, 261
 DARPA, 66
 datagrams
 fragmentation of, 109
 reassembly of, 109
 daylight savings time
 kernel, 30
 demultiplexing
 TCP/IP, 93
 device drivers
 Consulting Services, 47
 courses, 56
 device addresses, 48
 phone support, 51
 references, 57
 third party, 53
 device names
 SunOS installation, 63
 devices
 ones present, 163
 disk
 combinations with controllers, 61, 62
 determining configurations, 59
 enlarging procedure, 39
 enlarging SunIPC, 39
 dispatching
 procedures, 15
 DMA, 47
 DoD, 66
 domain system
 Internet, 103
 drivers
 courses, 56
 references, 57
 third party, 53
 DST, 30
 Australia, 30
 Europe, 30
 rules table, 31
 DVMA, 47

E

education
 courses, 26
 SunOS courses, 65
 Educational Services
 courses, 26

email
 Sun Education, 26
 Ethernet, 96
 back-to-back packets, 79
 buffer, 79
 controller, 79
 header, 96
 throughput, 80, 81
 experiment
 devices present, 163

F

files
 after power failures, 77
 fragmentation
 datagrams, 109
 frame buffers
 with screendump, 37
 ftime, 30
 FTP, 86

G

gateway, 66
 gateways, 106
 gettimeofday, 30
 GMT, 30

H

Hackers' Corner
 devices present, 163
 SunView, 261
 hardware
 color frame buffers, 140
 headers
 IP, 95
 octets, 91
 overview, 93
 history
 use of, 78
 hotline
 procedures, 15
 use of, 11
hotline@sun.COM
 reporting bugs, 13
 hotlines
 world, 7, 198

I

I/O
 sockets, 126
 ICMP, 102
 images
 converting to monochrome, 37
 installation
 SunOS, 63
 Intercon
 hotline, 7, 198
 Internet
 addresses, 107
 domain system, 103
 protocols, 85

IP, 85

headers, 95

K

kernel

booting specific, 76
daylight savings time, 30
time zones, 29**L**

labels

pedestal, 59

layering

mail, 91

level 1

network hardware, 123

level 2

network hardware, 123

localtime, 31

M

mail, 87

aliases, 155
formats, 157
layering, 91
pitfalls, 157
routing, 105

manuals

proprietary, 50

maps

color, 140
YP, 34

mask

address, 67

monitors

high-resolution, 37

MS-DOS, 39

N

networks

carrier sense, 114
collision detection, 115
Ethernet theory, 114
hardware problems, 122
performance of, 118
Q & A, 124
thin Ethernet, 122

NFS, 88

O

octets

TCP/IP headers, 91

out-of-band data

sockets, 126

P

packets, 96

back-to-back, 79

pedestal

information, 59

Personal AnswerLine, 9

port number

assignment of, 75

power failures

diskless workstations, 77

printing

images, 36

Prism

windows, 151

procedure

enlarging SunIPC disk, 39
hotline, 15

products

release levels, 6, 197

PROM monitor

using boot, 73

proprietary manuals, 50

R*Read This First*

purpose, 18

reassembly

datagrams, 109

references

device drivers, 57

release level

SunOS, 17

releases

software products, 6, 197

reporting bugs, 13

routing

mail, 105

RTF

purpose, 18

Rutgers University, 85

S

screendump, 36

color windows, 152

screenload, 37

server

stream socket, 127

shoobox

disk labels, 60

SIGIO, 126

SIGPIPE

server, 127

SIGQUIT

server, 127

SIGURG, 126

SMTP

application example, 100

sockets

example programs, 127
out-of-band data, 126, 133
programming examples, 126
servers, 127
well-known, 97

SPARC

with Sys4-3.2, 205

specials

CSD Consulting, 51

specials, continued
 device drivers, 51
specific kernel
 booting, 76
STB
 duplication of, 8, 199
subnets
 address mask, 67
 broadcasting, 107
 definition, 66
 enabling, 69
 Exterior Gateway Protocol, 66
 limitations, 68
 subnetting, 66
Sun Education
 device driver course, 56
 SunOS courses, 65
sun/hotline
 reporting bugs, 13
 use of, 11
sun/stb-editor, 8, 155, 199, 255
sun/sunbugs
 reporting bugs, 13
suncustomer-training
 Sun Education, 26
sunbugs@sun.COM
 reporting bugs, 13
SunCGI, 144
SunCore, 146
 printing images, 36
SunIPC
 enlarging disk, 39
SunOS
 determining release of, 17
 installation, 63
suntools
 frame buffers, 141
SunView
 color frame buffers, 142
 Hackers' Corner, 261
 under Sys4-3.2, 205
switcher (1)
 colormaps, 151
Sys4-3.2
 announcement, 205
 binary compatibility, 207
 hardware support, 206
 software configurations, 207

T

tables
 software release levels, 6, 197
tape drives
 SunOS installation, 63
TCP, 85
 sockets, 130
TCP/IP
 demultiplexing, 93
 references, 110
TELNET, 86
thin Ethernet

thin Ethernet, continued
 specification, 122
throughput
 Ethernet, 80, 81
time zones
 TZ, 29
 uucico, 30
training
 Sun Education, 26
TZ, 29
 DST rules table, 31

U

UDP, 102
update, 77
USA-4-SUN
 use of, 12, 15
USAC
 feedback, 10
utilities
 yellow pages, 34
uucico
 time zones, 30

W

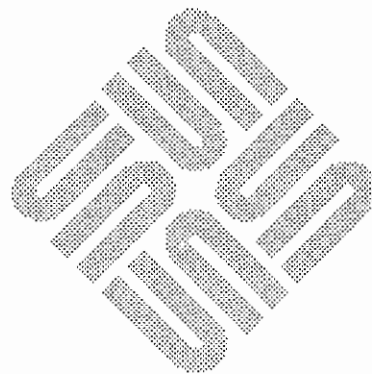
well-known sockets, 97
windows, 140
 color frame buffers, 141
 Prism, 151
 world hotlines, 7, 198

Y

yellow pages, 32
 installation, 33
 mail aliases, 155
 utilities list, 34
YP, 32
 clients, 32
 domains, 33
 installation, 33
 maps, 34
 master server, 32
 rpc, 34
 server maps, 32
 slave servers, 32
 utilities list, 34
ypbind, 32
ypserv, 32

Revision History

<i>Revision</i>	<i>Date</i>	<i>Comments</i>
FINAL	February 1988	Second issue of the 1988 Software Technical Bulletin, developed by Software Information Services (SIS), Customer Services Division (CSD).







Bulk Rate
U.S. Postage
PAID
Permit No. 515
Mountain View, CA

Corporate Headquarters

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
TLX 287815

For U.S. Sales Office

locations, call:
800 821-4643
In CA: 800 821-4642

European Headquarters

Sun Microsystems Europe, Inc.
Sun House
31-41 Pembroke Broadway
Camberley
Surrey GU15 3XD
England
0276 62111
TLX 859017

Australia: 61-2-436-4699

Canada: 416 477-6745

France: (1) 46 30 23 24

Germany: (089) 95094-0

Japan: (03) 221-7021

The Netherlands: 02155 24888

UK: 0276 62111

Europe, Middle East, and Africa,

call European Headquarters:
0276 62111

Elsewhere in the world, call

Corporate Headquarters:

415 960-1300
Intercontinental Sales

