



Software Technical Bulletin
December 1987

Software Information Services



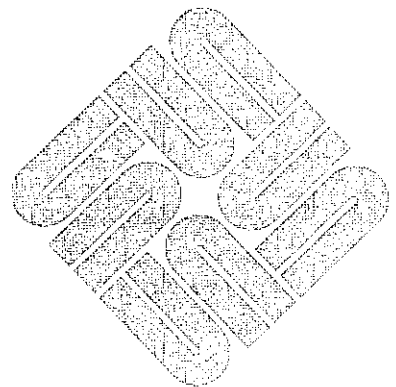
Part Number 812-8701-11
Issue 1987 - 11
December 1987





Software Technical Bulletin
December 1987

Software Information Services



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*. Customers who have technical questions about topics in the Bulletin should call Sun Customer Software Services AnswerLine at **800 USA-4-SUN**.

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 TransScript 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 © 1987 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.

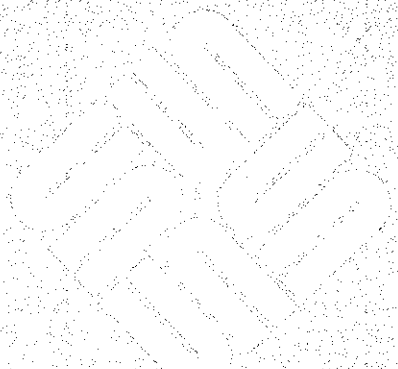
Contents

Section 1 NOTES & COMMENTS	975
Editor's Notes	975
Software Release Levels	977
World Hotlines	979
Errata	980
Section 2 ARTICLES	983
Using adb	983
Missing lost+found	985
SunOS Release 3.5	987
SunTrac Release 1.0	1000
Section 3 STB SHORT SUBJECTS	1007
Using boot	1007
Super Eagle Disks	1009
SunIngres 5.0	1010
Section 4 IN DEPTH	1013
Passing Commons	1013
Section 5 QUESTIONS, ANSWERS, HINTS, AND TIPS	1023
Q&A, and Tip of the Month	1023
Section 6 THE HACKERS' CORNER	1031
Porting SunView	1031
Section 7 CUMULATIVE INDEX: 1987	1049



NOTES & COMMENTS

NOTES & COMMENTS	975
Editor's Notes	975
Software Release Levels	977
World Hotlines	979
Errata	980





NOTES & COMMENTS

Editor's Notes

Editor's Notes

The December editor's notes for the Software Technical Bulletin (STB) include the current Sun software products and release levels table, a note on STB mailing, and a reminder to first use the AnswerLine number when calling for support. Finally, in **The Hackers' Corner**, this issue includes three example programs that illustrate ways to process event-driven input when running SunView.

Current Sun Software Products and Release Levels Table

The December Software Technical Bulletin (STB) includes the current version table. The current release level is shown for each product.

Use this table along with STB articles that appear for a particular product. You can then better determine what your software needs are, what functions are available in a new release, and whether the release you are using is down-level from the most current product release.

STB Mailing

The Customer Service Division (CSD) of Sun Microsystems, Inc. is putting new bulk mailing procedures into place to ensure proper tracking, sorting, and mailing of STB issues.

The transition to the new procedures and implementation of United States Postal Service regulations has caused a delay in the availability of some STB issues.

Thank you for your patience in this regard.

Call the AnswerLine First

Please use your **1-800-USA-4-SUN** AnswerLine first when calling for software support. This number allows Sun to dispatch your call and determine necessary billing information, based on your warranty or contract status, prior to a response from the appropriate United States Answer Center (USAC) support group.

USAC looks forward to answering your questions, but can do so only after a necessary service order number is generated by the dispatcher. Please refer to the article entitled 'Using USA-4-SUN' on page 567 of the September 1987 STB for details on dispatching, contract issues, and billing procedures that begin with your initial call to your AnswerLine.

World Hotlines Table

For Sun customers outside the United States, please call your local support group and follow the local software support procedures.

For your convenience, a table containing service hotlines around the world now appears monthly in the STB, beginning with this issue. Look for the world hotlines table in the Notes and Comments section each month.

The Hackers' Corner

This month's Hackers' Corner includes three example programs to illustrate how event-driven input is processed when running SunView.

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 November 20, 1987

Product Name	Current Release
SunOS	3.4
Cross Compilers	2.0
SunLink BSC3270	4.0
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	4.0
SunLink X.25	5.0
Sun FORTRAN	1.0
SunPro	2.0
NeWS	1.0
Sun Common Lisp-D	2.1
Sun Common Lisp-E	1.1
Modula-2	1.0
SunAlis	2.1
SunGKS	2.1
SunINGRES	5.0
SunSimplify	1.0
SunUNIFY	2.0
Transcript	2.0
SunIPC	1.1
PC-NFS	2.0
SunTrac	1.0

**Current Sun Software
Products and Release Levels**

The table appearing above contains a list 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 table below to determine whether your release is the current release level.

This table appears monthly in the STB for your convenience.

Sun FORTRAN Note

Please note that Sun FORTRAN is a value-added product that supports VMS extensions to the $\epsilon 77$ compiler, which is automatically included with SunOS release 3.4.


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

Errata

Errata

Please enter the corrections shown below into the appropriate articles.

European Hotlines

The 'European Hotlines' note on page 562 in Section 1 of the September 1987 STB needs updating. Please delete the European service hotline shown for Germany and replace it with the new Germany hotline (49)89/95094-321.

Client UNIX Status

Enter the corrections listed below into the 'Client UNIX Status' article found on pages 577-579 in Section 2 of the September 1987 STB.

1. Subheading `ping`, page 577

In line 1 of the first paragraph under this subheading, delete `imc echo` and add `icmp echo`.

In line 2 of the same paragraph, delete 'Inter-Process' and add 'Internet Protocol'.

2. Subheading `rpc.rstatd`, page 578

All references to '`rstat`' should refer to '`rstatd(8C)`'. All references to '`rpc.rstat`' should refer to '`rstat(3R)`'.

3. Subheading `pmap_rmtcall`, page 578

In line 4 of the first paragraph under this subheading, delete 'a user interface' and add 'an rpc library call'.

At the end of this section, add the following sentence: 'The user level program which allows you to determine if particular rpc daemons are running is '`rpcinfo(8)`'.'

ARTICLES

ARTICLES	983
Using adb	983
Missing lost+found	985
SunOS Release 3.5	987
SunTrac Release 1.0	1000

UNCLASSIFIED



ARTICLES

Using adb

Using adb: Determining
Your nd Server

You can use `adb` as an interactive, general-purpose debugger to conveniently determine which server is your Network Disk (nd) server.

Running `adb` on your `/vmunix` returns the storage address where the needed internet address information is located. You then use that information to determine your nd server internet address. After converting the address to a decimal representation, using the `ypmatch` command lets you determine your nd server machine name.

The `adb` command displays the nd server internet number in hexadecimal notation. An example is shown below. Note that the commands and values you enter are shown in bold.

```

machine% adb -k /vmunix /dev/mem <return>
sbr f068464 slr 649
physmem lfe
nd+708/X <return>
_nd+0x708:      c0090437
0t0xc0=D <return>
                192
0t0x09=D <return>
                9
0t0x04=D <return>
                4
0t0x37=D <return>
                55
^D <return>
machine%

```

The second and third lines are `adb` messages and can be ignored. Entering '`nd+708/X`' causes two values to appear. The first value is the address requested and the second portion is the value stored at that address. This value is the hexadecimal representation of your nd server's internet address.

Hexadecimal-to-Decimal Conversion

Continuing through the example shown above, you now need to convert the hexadecimal representation of the nd server internet address to a decimal format. Do this by entering the commands as shown.

This example yields a decimal representation of 192.9.4.55 for the nd server internet address.

Determining Your nd Server Machine Name

You can now determine your nd server machine name since you know the server's internet address. If you are not on yellow pages, the nd server name and internet address are found in your `/etc/hosts` file.

If you are on yellow pages, determine your nd server machine name by using the `ypmatch` command as shown below.

```
machine% ypmatch 192.9.4.55 hosts.byaddr
192.9.4.55      fredonia      # Department ND server
machine%
```


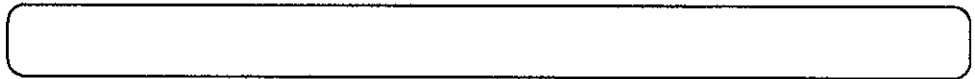
Please note that you must substitute your nd server internet address for the address used in the example above.

References for Further Information

adb(1)

Refer to chapter 4, 'adb Tutorial', in the manual *Debugging Tools for the Sun Workstation*, part number 800-1325, for details on how to use `adb`.

Related commands include *cc(1V)*, *dbx(1)*, *kadb(1)*, *ptrace(2)*, *ILa.out(5)*, *mem(4S)*, and *core(5)*.

 **Missing** lost+found**Missing** lost+found

This article contains information on a problem described in Bug Reference Number 1001492, and an available workaround.

mkfs(8) is the command that is run to initialize a new disk partition for a filesystem. The directory `/usr/lost+found` should be created by *mkfs(8)* as part of this process for later use as needed by *fsck(8)*.


The Problem

The problem is that setup for all SunOS releases 3.x removes the directory `/usr/lost+found` on standalone systems. This directory must be recreated manually.

The Workaround

Use the script contained in this article to manually create the directory `/usr/lost+found`. Follow the steps shown below.

1. Log in as root using `su`, the set user command.
2. Using `cd`, change directory to the root directory of the file system that is missing its `/usr/lost+found` directory.
3. Run the `mklost+found` shell script appearing below.

 **Reference for Further Information**

Refer to *fsck(8)* for more detailed information on the `/usr/lost+found` directory.

The Shell Script

The shell script appears on the following page.

```
#!/bin/csh -f

#####
#
# Shell script 'mklost+found'
#
# Creates a lost+found directory of the correct size
#
#####

rm -rf lost+found
mkdir lost+found
chmod 755 lost+found
chown root lost+found
chgrp wheel lost+found

cd lost+found

touch {0,1,2,3,4,5}{0,1,2,3,4,5,6,7,8,9}{0,1,2,3,4,5,6,7,8,9}\
6{0,1,2,3,4,5}{0,1,2,3,4,5,6,7,8,9}
rm [0-6]*

ls -ld . | grep -s 8192
switch ($status)
case 0:
    echo "${0}: lost+found directory created"
    exit (0)
default:
    echo "${0}: lost+found directory created but size is incorrect"
    exit (1)
endsw
```

SunOS Release 3.5

SunOS Release 3.5

This article is a brief overview of Sun Operating System (SunOS) Release 3.5, which will be shipped with all new Sun-2 and Sun-3 orders, and provided free (upon request) to all existing Sun workstation users holding current software support contracts.

Introduction

SunOS Release 3.5 includes the functions and features in Releases 3.3 and 3.4, and is upwardly compatible with Releases 3.2, 3.3, and 3.4. Thus, any program developed to run under these previous releases will run properly under Release 3.5.

SunOS Release 3.5 incorporates the following:

- Support for new hardware products
- High-priority bug fixes
- Reduced media set

Each of these is discussed below.

New Hardware Product Support

SunOS Release 3.5 supports the following new hardware products:

- Sun-3/60 Desktop Workstation
- Sun-3 Eurocard Board (3E)
- Double buffering capability

This support provides a Release 3.X base for new hardware products, thus allowing Sun customers greater flexibility in determining when to adopt future operating system releases in order to upgrade system hardware capabilities.

SunOS Release 3.5 eliminates the need for special support tapes, by consolidating support for the Sun-3/60 and 3E products. Additionally, the double buffering support will provide higher quality rendering of graphics images on future Sun graphics workstations. To utilize double buffered graphics images, additional code is provided by SunOS Release 3.5.

**Bug Fixes Included in SunOS
Release 3.5**

SunOS Release 3.5 incorporates all bug fixes from Release 3.4.1 and Release 3.4.2, in addition to new fixes since Release 3.4.2. These bugs are summarized by reference number, release(s) in which the bug occurred, and a brief one-line synopsis, as follows.

Release 3.4.1 Bug Fixes

Reference Number: 1004642

Release: 3.4 beta 3

Synopsis: screenblank allows the -k and -m options while in suntools.

Reference Number: 1003572

Release: 3.2

Synopsis: Bad inquire_cell_array and inquire_pixel_array name argument.

Reference Number: 1003687

Release: 3.2

Synopsis: The cgi mouse cursor is always visible.

Reference Number: 1004825

Release: 3.4 beta 3

Synopsis: -l cgi requires -lsuntool to compile a cgi program.

Reference Number: 1005251

Release: 3.4

Synopsis: close_cgi_pw() fails if no viewsurface is active.

Reference Number: 1003864

Release: 3.2

Synopsis: The crosshair cursor does not work when CANVAS_FAST_MONO is used.

Reference Number: 1004500

Release: 3.1, 3.2 (Sun3 fix only)

Synopsis: A program compiled using -f fpa causes an FPA KERNEL BUS ERROR to occur.

Reference Number: 1003023

Release: 3.2

Synopsis: The `fsck: HOLD BAD BLOCK` message is undocumented.**Reference Number: 1005131**

Release: 3.2

Synopsis: The resolver has the wrong loopback address.

Reference Number: 1003151

Release: 3.2

Synopsis: `make` does not always build the objects that it should.**Reference Number: 1004791**

Release: 3.4

Synopsis: `ping` says machines are up even when they are not.**Reference Number: 1004074**

Release: 3.2

Synopsis: `lprm` causes line printer daemon to disappear.**Reference Number: 1005140**

Release: 3.2

Synopsis: A `rexed` race condition occurs when mounting in `/tmp`.**Reference Number: 1004639**

Release: 3.2, 3.4 beta

Synopsis: Emulex SCSI tape controller and DocuPro page scanner do not work together correctly on the SCSI bus

Reference Number: 1005042

Release: 3.2

Synopsis: Yellow Page alias must use primary host names.

Reference Number: 1003135

Release: 3.2

Synopsis: `panic: mfree` occurs with `AF_UNIX SOCK_STREAM` out-of-band (OOB) data.

Reference Number: 1000895

Release: 1.1

Synopsis: Transformation of a Suncore segment containing text is not clipped.

Reference Number: 1004898

Release: 3.4

Synopsis: The `install_sunpro` script fails for all configurations.

Reference Number: 1004731

Release: 3.2

Synopsis: `termcap` entry for `TERM=wy` breaks `initscr()`.

Release 3.4.2 Bug Fixes

Reference Number: 1003647

Release: 3.2

Synopsis: Lexically recursive `#includes` confuse `dbx`.

Reference Number: 1004996

Release: 3.4, 3.2

Synopsis: `dbx` shows segmentation violation while stepping.

Reference Number: 1005466

Synopsis: `sysdiag`'s `sptest` fails w/ `/dev/tty[a,b]`; does not respond.

Reference Number: 1005360

Release: 3.4, 3.3

Synopsis: SCSI disk driver hangs when `ACB4000` reports write fault.

Reference Number: 1005363

Release: 3.4, 3.3

Synopsis: Some SCSI MD21 (141 MB) errors cause system hang.

Reference Number: 1006127

Release: 3.4, 3.3

Synopsis: Ethernet problems induced by bad ICMP address mask reply.

Reference Number: 1005930

Release: 3.3, 2.X, 1.X

Synopsis: `physio` bug causes `writew(2V)` failure.**Reference Number: 1001069**

Release: 1.X, 2.X

Synopsis: Bug in `physio` breaks `readv`.**Reference Number: 1006165**

Release: 3.4

Synopsis: `sysdiag`'s `softfp` and `mc68881` tests core dump on an illegal instruction.**Reference Number: 1004863**

Release: 3.2

Synopsis: `GP1_PR_PGON_TEX` problem.**Reference Number: 1004984**

Release: 3.2

Synopsis: `GP1_PR_ROP_TEX` semantics are wrong for a 1-bit deep `src`.**Reference Number: 1005359**

Release: 3.4, 3.2

Synopsis: `pw_line` and `pw_polyline` do not draw a vector from left to right when the starting point has a negative 'x' coordinate.**Reference Number: 1004336**

Release: 3.4, 3.2

Synopsis: `lockf()` very slow.**Reference Number: 1003885**

Release: 3.2

Synopsis: `look` may dump core on long lines.**Reference Number: 1004765**

Release: 3.3

Synopsis: Subnet broadcast address computed incorrectly.

Reference Number: 1005489

Synopsis: NFS attribute cache functions incorrectly.

Reference Number: 1004739

Release: 3.2

Synopsis: `rpc.lockd` fails to free, thus using excess memory.

Reference Number: 1003207

Release: 3.2

Synopsis: SCCS uses delta times for `diffs`.

Reference Number: 1005438

Release: 3.2

Synopsis: SCCS `deledit` duplicates random lines in a file.

Reference Number: 1005366

Release: 3.3

Synopsis: System returns `panic: Bus error` message when using `ttya` with a configured kernel on a system with a SCSI3 host adapter.

Reference Number: 1006154

Release: 3.4

Synopsis: System is flooded with `zs` interrupts on `synca/b` transitions.

Reference Number: 1004598

Synopsis: `make` does not handle square bracket characters in target filenames.

Reference Number: various sunpro make bugs.

Synopsis: Various unnumbered fixes as described below.

Descriptions:

- 1) `make` no longer dumps core if the source needed to build a library member does not exist; instead reports "Don't know how to build x".
- 2) Fixed the `-k` option so that it works for lists of targets given on the `make` command line.

3) Remove the `.make.state` lock file if make is interrupted.

4) Use the `varargs` mechanism for the error routines.

Reference Number: 1004559

Release: 3.4, 3.2

Synopsis: UNIX hangs while booting if `xt` controller has on-line drive.

Reference Number: 1006132

Release: 3.4

Synopsis: TCP/IP file transfer using `ftp` hangs/stops when using 3.4.

**Bug Fixes New to SunOS
Release 3.5**

Reference Number: 1001271

Synopsis: `ptrace` interaction with interrupting slow system calls.

Reference Number: 1002675

Release: 3.2, 3.0, 2.0

Synopsis: Driver error message is 'cryptic' (MTI).

Reference Number: 1002968

Release: 3.2

Synopsis: Incorrect comment in `/usr/include/sys/buf.h`.

Reference Number: 1004165

Release: 3.2

Synopsis: Setting raw mode under `bk (4)` line discipline panics.

Reference Number: 1004195

Release: 3.4beta

Synopsis: `vi` breaks on `!!` command with long output.

Reference Number: 1004200

Release: 3.2

Synopsis: SCSI tape drivers error handling inconsistent.

Reference Number: 1004256

Release: 3.2, 3.0
Synopsis: `ld -r` confuses `dbx`.

Reference Number: 1004323

Release: 3.2
Synopsis: Re-debugging prog w aborted open of pipe crashes system.

Reference Number: 1004364

Release: 3.2
Synopsis: `overwrite` in 4.3 `-lcurses` drops core.

Reference Number: 1004503

Release: 3.2
Synopsis: Serial port device driver panic.

Reference Number: 1004577

Release: 3.2
Synopsis: `printf` padding strings with leading zeros is broken.

Reference Number: 1004768

Release: 3.3, 3.2
Synopsis: `Maxusers` causes `sys pt` too small message when booting.

Reference Number: 1004840

Release: 3.2
Synopsis: `RSTAT(3R)` always returns 0 in `statstime.if_opackets`

Reference Number: 1005063

Release: 3.2
Synopsis: `ld -A` produces bogus symbol tables.

Reference Number: 1005068

Release: 3.2
Synopsis: Misspelled error message in `trap.c`.

Reference Number: 1005165

Release: 3.2

Synopsis: 3.X kernel can't handle failure of TOD chip on 3/50s.

Reference Number: 1005241

Release: 3.2

Synopsis: Boot from 1/2" tape (tm/cdc) on 3/260 hangs system.

Reference Number: 1005242

Release: 3.4

Synopsis: Upgrade doesn't install symlinks for screenld.

Reference Number: 1005252

Release: 3.4

Synopsis: tektool: PROPS key no longer works.

Reference Number: 1005253

Release: 3.4

Synopsis: tektool: menu missing items and no longer in walking style.

Reference Number: 1005260

Release: 3.2

Synopsis: tektool: certain data streams causes core dump.

Reference Number: 1005310

Release: 3.4

Synopsis: cgtwo driver watchdogs if no vector is specified

Reference Number: 1005381

Synopsis: rasfilter8to1.lg has wrong filename suffix, should be '.1'

Reference Number: 1005391

Release: 3.4, 3.3

Synopsis: ^C in diag using SCSI-3 host adaptor causes error message.

Reference Number: 1005485

Release: 3.4
Synopsis: TCP performance problem.

Reference Number: 1005580

Synopsis: `sysdiag` puts logs in `/usr2`, filling the root partition.

Reference Number: 1005812

Release: 3.4, 3.2
Synopsis: Programs whose inodes exceed the 512 Mb point will core dump with a segmentation violation.

Reference Number: 1005983

Release: 3.2
Synopsis: `PR_PGON_TEX` does not clip texture to screen properly on GP.

Reference Number: 1006055

Release: 3.4
Synopsis: Upgrade on 68010 does not upgrade `ndll`.

Reference Number: 1006093

Release: 3.2
Synopsis: `pr_load` may behave badly in case of error.

Reference Number: 1006103

Release: 3.2
Synopsis: `if.c` changes to support MCP and subnets.

Reference Number: 1006202

Release: 3.4
Synopsis: Some colormap updates fail on 3/60 color frame buffer.

Reference Number: 1005304

Release: 3.4, 3.2

Synopsis: Shell script crashes 3/200 series kernel.

Reference Number: 1006668

Release: 3.4

Synopsis: `pr_polypoint` on GP may draw first point incorrectly.**Reference Number: 1006164**

Release: 3.4

Synopsis: `sysdiag` assumes you are always on console.**Reference Number: 1003351**

Release: 3.4, 3.2

Synopsis: `pw_line()` draws in window space instead of canvas space.**Reference Number: 1006123**

Release: 3.4, 3.2

Synopsis: `pw_line()` draws incorrectly when `pw_batch_on()`.**Reference Number: 1006690**Synopsis: 2.59 `sysdiag` disables disk testing in systems with GP/GB.**Reference Number: 1006255**

Release: 3.4, 3.3

Synopsis: network routing daemon `in.routed` dies periodically with 3.3 and 3.4.**Reference Number: 1006729**

Release: 3.4

Synopsis: routing daemon `in.routed` sometimes uses wrong interface.

Reference Number: 1004864

Release: 3.4, 3.2

Synopsis: last incorrectly reports 'still logged in'.

Reference Number: (N/A)

Synopsis: fsck silently fails to fix partially truncated inodes.

Reduced Media Set

The entire system software media set is contained on five (5) tapes, thus reducing the number of tapes necessary for operating system installation.

Upgrade Considerations

The upgrade path on the SunOS Release 3.5 distribution set can be used to upgrade from either SunOS Release 3.2 or Release 3.4.

Obtaining SunOS Release 3.5

SunOS Release 3.5 consists of a full distribution set of five (5) tapes, a Right-To-Use (RTU) license, and documentation set for domestic and international customers, as shown below.

Order Number	Format	Includes	License Type
UPSYS2-01F	68010	1/4" tape, RTU, documentation	Domestic
UPSYS2-02F	68010	1/2" tape, RTU, documentation	Domestic
UPSYS2-03F	68010	1/4" tape, RTU, documentation	Export
UPSYS2-04F	68010	1/2" tape, RTU, documentation	Export
UPSYS3-01F	68020	1/4" tape, RTU, documentation	Domestic
UPSYS3-02F	68020	1/2" tape, RTU, documentation	Domestic
UPSYS3-03F	68020	1/4" tape, RTU, documentation	Export
UPSYS3-04F	68020	1/2" tape, RTU, documentation	Export
UPSYS-00F		RTU license only	

Workstations which are covered under a current software support contract will be provided with the full distribution set at no additional charge. This release will not be automatically shipped, but is available upon customer request. To request the SunOS Release 3.5 set, call (800) USA-4-SUN. This number can also be used by contract customers who would like to obtain only the Release 3.5 documentation set.

SunTrac Release 1.0



SunTrac Release 1.0

This article is an overview of SunTrac Release 1.0, Sun Microsystems' new graphics-based project management software that can be used on Sun-2, Sun-3, and Sun-4 workstations running Sun Operating System (SunOS) Release 3.2 or higher.

Introduction

SunTrac Release 1.0 is a graphics-based planning and scheduling project management package that has been developed for the SunView window environment. SunTrac combines Gantt charts showing task durations, PERT (Program Evaluation and Review Technique) charts showing the entire project broken down into smaller tasks, and Critical Path analysis techniques with a unique risk assessment methodology. The SunTrac system accommodates all levels of project participation by building hierarchical project models to offer a full range of detail, from individual task to summary-level overviews.

SunTrac's primary objective is to provide all levels of technical management with the capability to accurately create, analyze, and communicate the planning, scheduling, and controlling functions that are necessary to ensure the completion of all project objectives within specified schedule and budget limitations. To achieve this objective, SunTrac utilizes a new algorithmic technique called Trac (Total Risk Analysis Calculation) to address the effects of uncertainty in project cost and schedule milestones. Trac analyzes the risk factors with regard to the project schedule, thus aiding the user in directing ongoing work appropriately.

Instead of focusing on a single critical path, SunTrac assigns a criticality index to each of the project's activities. This criticality index has a value between zero (least critical) and one (most critical).

SunTrac Project Management Tools

In addition to Trac, the SunTrac project management system incorporates the following tools.

Sketch (Network Diagram Editor): Sketch is the graphics editor used to create PERT network diagrams and associated data. Sketch creates ASCII files for use as input to the Trac analysis program. Sketch also accepts output produced by other SunTrac tools, and incorporates this data back into the PERT diagram. Sketch is also used to print the network diagram on a laser printer.

Level (Interactive/Automatic Resource Leveling Tool): Level simultaneously displays a Gantt chart of task durations with a resource profile to graphically represent resource usage, schedule activity, and control activity.

Assign (Optimal Overtime Allocation Tool): Assign is used to create, display, and select an overtime allocation plan. Assign plans the least-cost reduction in a project schedule, keyed to a user-selected productivity factor. The data obtained from the Assign overtime allocation plan can then be merged with the Sketch input data.

Profile (Graphic Summary): Profile SunTrac's graphing tool, displays, analyzes, and prints staffing level and cost profiles for the network and data selected from the most recent Trac execution.

Report (Tabular Data Output): Report reviews, sorts, and extracts detailed tabular data for each activity, diagram, or subnetwork of a project. The data can be sorted on any of the data column headings. Report takes its input from four temporary files created by Trac.

HelpTrac (On-Line Help): HelpTrac is SunTrac's on-line help facility, which provides systematic application development and assistance for all of the SunTrac tools, in addition to listings of error messages and explanations for all SunTrac applications.

Traditional Analysis Methods and the Trac Algorithm

With traditional PERT analysis, optimistic and pessimistic estimates of task completion times are considered in calculations, but no attempt is made to use these estimates in a probabilistic analysis of the overall project completion time. Another limitation of PERT analysis is that it ignores all activities that are not on the critical path, as well as the stochastic (involving a random variable) nature of their respective completion times.

With the Monte Carlo simulation approach, a time is drawn from each of the activity distributions, a total project time is generated as in a deterministic model, then the process is repeated a large number of times, in order to determine a reliable sample mean and variance. This approach has as its main disadvantage the computer processing time required to form a statistically meaningful sample.

The Trac algorithm for stochastic analysis takes the data entered into Sketch (the input metrics) and calculates all of the essential output metrics necessary for other SunTrac programs to produce their respective reports and displays, such as Gantt charts, network diagrams, and tables and graphs of metric data. The Trac algorithm models the time to complete each task as a probability distribution by using the optimistic and pessimistic completion times as deviations from a mean completion time, where the optimistic and pessimistic estimates are actually interpreted as three standard deviations from the mean.

While the concept of using probability distributions instead of fixed times in PERT analysis is not new, it has been difficult to find a useful mathematical model, because of the complications involved in calculating probabilities for networks that contain many parallel paths. In contrast to these methods, the Trac algorithm uses a piecewise linear model of the distribution to calculate project completion probabilities quickly enough to allow repeated "what-if" trials of project scheduling and staffing.

Trac reports several completion times (deterministic, optimistic, expected, and pessimistic) for the project being analyzed. The different completion dates result from different ways Trac calculates the schedule.

Deterministic completion time is based on the expected value of each activity. The expected value of each activity duration is the 50% probability duration, which is not necessarily the same as the most likely duration estimate. The deterministic project completion date is usually near the 30% completion date. Because the deterministic schedule uses only one calculated duration value for each activity, it has no uncertainty associated with it.

Optimistic completion time has the same meaning in reference to project completion as the optimistic estimate does in activity completion; that is, there is less than a 1% chance of completing sooner than this date.

Expected completion time is equivalent to the 50% completion time. The expected date is always later than or the same as the deterministic date. There is an even chance of completing the project by this date.

Pessimistic completion time has the same meaning in reference to project completion as the pessimistic estimate does in activity completion; that is, there is less than a 1% chance of completing later than this date.

Trac Summary Metrics

Some network summary metrics are also included in the Trac report to help the user compare two or more project plans. Most of the metrics are straightforward and easy to understand. A few required more detailed explanation, as follows.

Network Complexity is a measure of network cross-connectedness. It is calculated using the following formula.

$$(\text{number of arcs} - \text{number of nodes}) / \text{number of arcs}$$

As values of network complexity approach unity (+1), the network is considered to be increasingly complex; that is, the number of arcs is greater than the number of nodes. Very simple networks can have negative values; that is, the number of nodes is greater than the number of arcs.

Stochastic Complexity, another measure of network complexity, measures the degree of occurrence of near critical paths. Values of stochastic complexity range between zero and one. A zero represents the simple network, with no near-critical paths, and a one represents a network with maximum cross-

connectedness, where Network Complexity approaches +1 and all paths are critical. In a comparison of two paths, the lower complexity factor is more desirable.

Stochastic Density is a summary measure of the slack in a project schedule, and is calculated using the following formula.

$$A / (A + B)$$

where *A* is the sum of all expected durations, and *B* is the sum of all free slack. The value falls between zero and one. A zero represents no duration, or infinite slack; a one represents no project slack. Lower stochastic density is desirable.

Plans with lower density are easier to manage for the following reasons:

- Project schedules are easier to shorten
- Staffing profiles are easier to level
- Recovery from unforeseen disasters is easier

Maximum Values for SunTrac Applications

Maximum values and limits for SunTrac applications are as follows.

Application	Maximum Value	Units
Sketch	250	nodes per diagram
	250	arcs per diagram
Trac	4000	nodes total per analysis
	4000	arcs total per analysis
	200	diagrams per analysis
Level	800	activities
Assign	800	activities
Profile	1800	days

Installation and Use Considerations

Refer to the *Software READ THIS FIRST* document provided with your copy of SunTrac Release 1.0 media for information relating to SunTrac installation and usage.

Required Risk Analysis for Department of Defense (DoD) Contractors

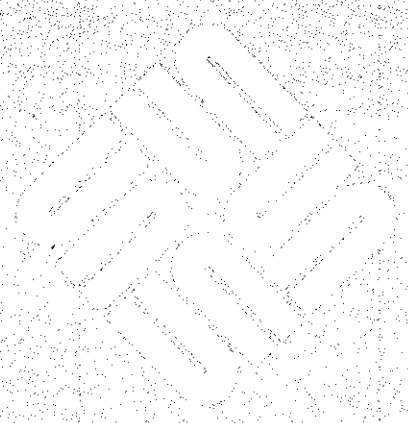
SunTrac can be used by organizations who contract with the Federal Government and must comply with Department of Defense Standard DoD-STD-2167 when producing the required Risk Management Procedures in software development plans. Refer to Appendix A of the *SunTrac Reference Guide*, part number 800-2059, for complete information.

**Additional Swap Space
Required**

The Trac process requires 3 MB of free swap space in addition to the swap space currently being used.

STB SHORT SUBJECTS

STB SHORT SUBJECTS	1007
Using boot	1007
Super Eagle Disks	1009
SunIngres 5.0	1010





STB SHORT SUBJECTS

Using `boot`

Using the `boot` Command from the PROM Monitor Prompt

Use the `boot` command to list the contents of the root directory when in the PROM monitor. This is particularly useful when `/vmunix` is corrupted or missing and you are looking for a backup version of the kernel.

An example of using `boot` from the PROM monitor prompt is shown below.

```
>b *
```

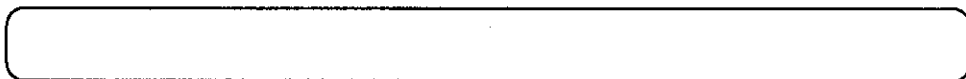
This will list all of the contents of the root partition. The sample result shown below is a typical listing for a Sun-3 server.

```
>b *
bin
boot
dev
etc
kadb
lib
lost+found
mnt
private
private.MC68020
pub
pub.MC68020
stand
sys
tftpboot
tmp
usr
usr.MC68020
vmunix
vmunix.gen
```

In this example, the backup version of the kernel is `vmunix.gen`.



Super Eagle Disks



Super Eagle Disk File System Sizes

Customers having the Super Eagle disk will find information in this article helpful to avoid core dumps and segmentation violation errors.

The Fujitsu 2361 Super Eagle is a 694Mb, 10-1/2" SMD disk drive using the Xylogics 451 controller.

The Problem

Some large programs fail with a segmentation fault and core dump. The bug ID reference number is 1005812.

This problem has the appearance of being intermittent, failing on some compilations and applications and not on others. A program will run in certain directories and not in others.

The problem has been observed on systems with Super Eagle drives.

The Cause

The cause has been identified as a limitation in the file system. If any partition is larger than 512Mb, programs located past the 512Mb point will not page correctly. Those programs stored on disk with disk block addresses exceeding this point will core dump with a segmentation violation.

The Workaround

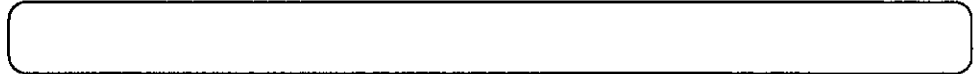
You can avoid this problem by reconfiguring your Super Eagle so that no partition is larger than 512Mb.

References for Further Information

Refer to the manual and article listed below for detailed information on disk drive formatting, labeling, and using `setup` to set file system sizes.

- *Installing UNIX on the Sun Workstation*, Section 3.6, 'Disk Overview and Philosophy', part number 800-1521
- *System Administration for the Sun Workstation*, Chapter 4, 'Disks and File Systems', part number 800-1323
- *Software Technical Bulletin*, November 1987, 'System and SunOS Installation Aids', page 775, part number 812-8701-10

SunIngres 5.0



Upgrading to SunIngres Release 5.0

For those customers upgrading from SunIngres release 3.0 to release 5.0, you may encounter three problems during the upgrade installation. This short subject contains problem descriptions and a suggested workaround.

The Problems Defined

The three main problems are described below.

□ *Permissions*

The permissions for `ingres/lib/*` are all 444. The old libraries are therefore not overwritten when the new tape is tarred.

□ *Root Ownership*

Two processes are owned by root and therefore cannot be overwritten. The processes are `ingres/bin/kill_ing` and `ingres/bin/ntproc`.

□ *Undeleted Binaries and Libraries*

The binaries and libraries that are no longer used in SunIngres release 5.0 are not deleted. The customer then has binaries and libraries taking up extra disk space unnecessarily.

The Workaround

The workaround for these problems is to delete `ingres/bin/*` and `ingres/lib*` prior to the upgrade installation. This will assure you that all of the libraries and binaries are the release 5.0 version.

CAUTION: Ensure that `ingres/data` and `ingres/files` are *not* deleted.

IN DEPTH

IN DEPTH 1013

Passing Commons 1013



Passing Commons

Passing FORTRAN Common Variables to C

This tutorial explains how to pass FORTRAN common variables to and from C programs.

Example of a Common Block

Let us start with a very simple example. Here is a FORTRAN program with a blank (unnamed) common variable.

```
common // hello
integer hello
print *, 'before initialization: hello = ',hello
hello = 9
print *, 'after initialization: hello = ',hello
end
```

When you compile this program using `f77 f.f` and run it, you will get the output shown below.

```
before initialization: hello = 0
after initialization: hello = 9
```

Passing a Common Block to a C Function

Now, suppose you want to pass the FORTRAN common to a C language subroutine. In this example, the C routine will change the value of `hello` and print it.

On the next page, let's see the FORTRAN program again with a call to a C routine and a print statement added.

```

common // hello
integer hello
print *, 'before initialization: hello = ',hello
hello = 9
print *, 'after initialization: hello = ',hello
call csub()
print *, 'after return from csub: hello = ',hello
end

```

The C function has the effect of changing the value of `hello`.

```

struct commonBLNK
{
    int hello;
} _BLNK__;

csub_()
{
    printf("csub:at the top: hello = %d\n", _BLNK__.hello);
    _BLNK__.hello = 7;
    printf("csub:after the value change: hello = %d\n", _BLNK__.hello);
}

```

When you compile and run these together using the command `f77 t.f c.c`, you get the following output.

```

before initialization: hello = 0
after initialization: hello = 9
csub:at the top: hello = 9
csub:after the value change: hello = 7
after return from csub: hello = 7

```

Appended Trailing Underscores

In line 5 of the called C routine, note that the subroutine name `csub` has an underscore (`_`) following it. Note also that `_BLNK__` in lines 4, 8, 9, and 10 has two trailing underscores. The internal name of the blank common is `_BLNK_`. However, all references to `_BLNK_` in the C subroutine require the additional trailing underscore.

Note that these underscores are required since the `f77` compiler appends a trailing underscore to all external names in FORTRAN programs. Refer to Section 4.5, 'Interprocedure Interface', of the *FORTRAN Programmer's Guide*, part number 800-1371, for further information on C and FORTRAN interfacing.

The structure type `commonBLNK` has as its members the variables in the common block. In this case it has the integer `hello`. The name of the actual structure is `_BLNK__`. Thus, in the C code you reference, the variables in a FORTRAN common as structure members. FORTRAN commons are global data. Therefore, the C structures that you declare must also be global.

Examples Using Naming Conventions

If your common has a name, use the same naming convention in your C subroutine. Substitute the common name for each occurrence of `_BLNK_`.

Here are our examples again. The common now has the name `greetings`.

The FORTRAN main program follows.

```
common /greetings/ hello
integer hello
print *, 'before initialization: hello = ',hello
hello = 9
print *, 'after initialization: hello = ',hello
call csub()
print *, 'after return from csub: hello = ',hello
end
```

The C subroutine is shown again below. Note that `greetings` has replaced the term `_BLNK_`.

```
struct commongreetings
{
    int hello;
} greetings_;
csub_()
{
    printf("csub:at the top: hello = %d\n", greetings_.hello);
    greetings_.hello = 7;
    printf("csub:after the value change: hello = %d\n", greetings_.hello);
}
```

You will see the following program output after compilation using the command `f77 t.f c.c`, and then executing the program.

```
before initialization: hello = 0
after initialization: hello = 9
csub:at the top: hello = 0
csub:after the value change: hello = 7
after return from csub: hello = 9
```

Note that `greetings` has one trailing underscore like `csub` and `_BLNK_`.

Complex Example

The following example shows a more complex common. This one has variables of several different types and sizes.

The FORTRAN code is shown on the next page.

```
common /greetings/ hello,shortone, string, pad, longone
integer hello
integer*2 shortone
character*3 string
character*3 pad
integer longone
print *, 'before initialization: hello = ',hello
print *, 'before initialization: shortone = ',shortone
print *, 'before initialization: string = ',string
print *, 'before initialization: pad = ',pad
print *, 'before initialization: longone = ',longone
hello = 9
shortone = 45
string = 'yz\0'
pad = 'pad'
longone = 167
print *, 'after initialization: hello = ',hello
print *, 'after initialization: shortone = ',shortone
print *, 'after initialization: string = ',string
print *, 'after initialization: pad = ',pad
print *, 'after initialization: longone = ',longone
call csub()
print *, 'after return from csub: hello = ',hello
print *, 'after return from csub: shortone = ',shortone
print *, 'after return from csub: string = ',string
print *, 'after return from csub: pad = ',pad
print *, 'after return from csub: longone = ',longone
end
```

The C subroutine code follows on the next page.

```
struct commongreetings_
{
    int hello;
    short shortone;
    char string[3];
    char pad[3];
    int longone;
} greetings_;
csub_()
{
    printf("csub:at the top: hello = %d\n", greetings_.hello);
    printf("csub:at the top: shortone = %d\n", greetings_.shortone);
    printf("csub:at the top: string = %s\n", greetings_.string);
    printf("csub:at the top: pad = %s\n", greetings_.pad);
    printf("csub:at the top: longone = %d\n", greetings_.longone);
    greetings_.hello = 7;
    greetings_.shortone = 15;
    greetings_.string[0] = 'd';
    greetings_.string[1] = 'e';
    greetings_.string[2] = '\0';
    greetings_.pad[0] = 'g';
    greetings_.pad[1] = 'h';
    greetings_.pad[2] = '\0';
    greetings_.longone = 67;
    printf("csub:after value change: hello = %d\n", greetings_.hello);
    printf("csub:after value change: shortone = %d\n", greetings_.shortone);
    printf("csub:after value change: string = %s\n", greetings_.string);
    printf("csub:after value change: pad = %s\n", greetings_.pad);
    printf("csub:after value change: longone = %d\n", greetings_.longone);
}
```

On the next page, the output shown results after compiling these programs.

```

before initialization: hello = 0
before initialization: shortone = 0
before initialization: string =
before initialization: pad =
before initialization: longone = 0
after initialization: hello = 9
after initialization: shortone = 45
after initialization: string = yz
after initialization: pad = pad
after initialization: longone = 167
csub:at the top: hello = 9
csub:at the top: shortone = 45
csub:at the top: string = yz
csub:at the top: pad = pad
csub:at the top: longone = 167
csub:after value change: hello = 7
csub:after value change: shortone = 15
csub:after value change: string = de
csub:after value change: pad = gh
csub:after value change: longone = 67
after return from csub: hello = 7
after return from csub: shortone = 15
after return from csub: string = de
after return from csub: pad = gh
after return from csub: longone = 67

```

Long Variables and Word-Boundaries

You should have no trouble creating C structures to match FORTRAN commons. FORTRAN is the more restrictive language for long variables being located on word-boundaries. However, if you are creating a FORTRAN common from a C structure that already exists, you may have to add some extra padding space to both the C structure and the FORTRAN common to align them with the FORTRAN word-boundary rules.

To see this, remove the declarations and all references to 'pad' from the above examples. The FORTRAN program will give you an error message when you compile it, but the C program will not.

C Main Programs with FORTRAN Subroutines

The examples up to this point show you how to call a C subroutine from FORTRAN and use I/O in both. However, if you have a C main program you will have to make a few minor changes as shown in the following paragraphs.

Here are our original two programs, but now reversed. The C program is the main routine, and the FORTRAN program is the subroutine.

The C code follows on the next page.

```

struct commonBLNK
{
    int hello;
} _BLNK__;

main()
{
    printf("main:before initialization: hello = %d\n", _BLNK__.hello);
    _BLNK__.hello = 9;
    printf("main:after initialization: hello = %d\n", _BLNK__.hello);
    fsub_();
    printf("main:after return from fsub: hello = %d\n", _BLNK__.hello);
}

```

The FORTRAN code appears below.

```

subroutine fsub
common // hello
integer hello
print *, 'fsub:at the top: hello = ',hello
hello = 7
print *, 'fsub:after the value change: hello = ',hello
end

```

But, look at what happens when you try to compile them. The results follow.

```

muse>> f77 t.f c.c
t.f:
t.f:
      fsub:
c.c:
Linking:
Undefined:
  _MAIN_
muse>>

```

You get this error message occurs because the link editor (ld) is trying to link in parts of the FORTRAN I/O subsystem to initialize it correctly. To resolve the problem, we create a 'dummy' FORTRAN main program and make the C main program an ordinary subroutine that looks and acts like a C main program, as shown in the next example.

'Dummy' FORTRAN Main Program Examples

Note that much of the added code you will see below in the modified C program is not needed for our small test case. The example shows all changes needed to make the typical C main program run properly. In particular, the example shows you how to incorporate command-line argument processing without changing most of the source code. For our simple case, you would only need to rename 'main'.

The modified C code appears below.

```

struct commonBLNK
{
    int hello;
} _BLNK_;

extern int xargc; /* allow access to argv and argc from a routine */
extern char **xargv; /* other than main */
/* xargc and xargv are declared and initialized */
/* in standard library code */

cmain_() /* rename main, remember the trailing "_" */
{
    int argc; /* declare argc and argv to use in the rest */
    char **argv; /* of your old main program */
    argc = xargc; /* initialize argc, and argv */
    argv = xargv; /* now argument processing can occur without
/* further modification to your old C main */

    printf("main:before initialization: hello = %d\n", _BLNK_.hello);
    _BLNK_.hello = 9;
    printf("main:after initialization: hello = %d\n", _BLNK_.hello);
    fsub_();
    printf("main:after return from fsub: hello = %d\n", _BLNK_.hello);
}

```

The modified FORTRAN code follows.

```

integer cmain,n
n = cmain()
end

subroutine fsub
common // hello
integer hello
print *, 'fsub:at the top: hello = ',hello
hello = 7
print *, 'fsub:after the value change: hello = ',hello
end

```

The only FORTRAN code change is the addition of the 'dummy' main program. We make `cmain` a function to obtain its return value, if any. We can call `exit` with this value to signal either normal or abnormal program completion.

Note that in `dbx`, you need to use the name `MAIN` to stop in the FORTRAN dummy main program. Use the name `cmain` to stop in the C main routine, since we have renamed it.

QUESTIONS, ANSWERS, HINTS, AND TIPS

QUESTIONS, ANSWERS, HINTS, AND TIPS 1023

Q&A, and Tip of the Month 1023



QUESTIONS, ANSWERS, HINTS, AND TIPS

Q&A, and Tip of the Month

Hints & Tips #9

This is the ninth in a continuing series of this column which I have created for two purposes.² 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!

Generic and Custom Kernels

One of the last steps to in the system installation process is configuring your system kernel. This is described in chapter 7, 'Configuring the System Kernel', in the *Installing UNIX on the Sun Workstation* manual, part number 800-1317.

Yet, many people neglect to do so. This important step should not be forgotten since it is an important part of making the most of your Sun workstation.

Since a generic kernel has to boot on an arbitrary configuration, every device driver for all devices supported by Sun is compiled into the kernel so the driver can be used if needed. This makes the kernel very large.

² This continuing column is submitted by Chuq Von Rospach, Customer Software Services.

When you boot your system, the kernel loads itself into memory. Any memory used by the kernel is unavailable for your user processes. Even if the drivers are not used, they are loaded into memory.

Configuring so that only the needed drivers for your system are compiled into your kernel can release up to 250k of memory. On a smaller system like a 4 megabyte Sun 3/50, this can mean the difference between a system slowed by a high paging rate and a system with reasonable performance.

Setting Maxusers

Another reason to configure a kernel is to properly size UNIX internal tables. Some data structures are stored in fixed-size tables. If maxusers is set too small the system will get spurious failures and error messages like 'text: table is full'. If you see this message, or similar messages for the 'proc' or 'inode' tables, then the kernel is overflowing its tables and the table sizes need to be expanded.

The size of these tables is controlled by the 'maxusers' variable in the kernel configuration file. This name is unfortunate, because the size of this value is not directly related to the number of users on the system, but to the loading and activity on the system. A system with a dozen people running the vi screen editor will perform happily with a smaller maxusers setting than the same system with a single user using a number of tools in the `suntools` environment.

The generic kernel shipped on the SunOS distribution tape has maxusers set to 4. For many system loads, this is too small. If you are using a bitmap and `suntools`, you should set maxusers to 8 or more.

One final hint on configuring a kernel. When you build your kernel configuration file, make sure you change the 'ident' line to something other than 'GENERIC'. This is a special ident that causes some special code to be compiled into the kernel code, that you do not need in a custom kernel. In one case, this custom code even causes problems. If you attempt to build a kernel with two swap spaces and a GENERIC ident, the kernel compilation will fail with an unresolved variable `_setconf`. The fix is to use an ident other than 'GENERIC'.

Finally, refer to the article 'Booting a Specific Kernel' on page 771 of the November 1987 STB. This article contains information on booting one of two available kernels in the case that you prefer to run a customized kernel for some applications.

Tip of the Month

This month's Tip is from Daniel Steinberg, and is a short C program that allows you to check whether some account or machine is in a netgroup. It is also interesting because it is a good practical example of how to program the yellow pages interface.

For more information on yellow pages netgroups, refer to Chapter 2, 'Sun Network Services', in the *System Administration for the Sun Workstation* manual, part number 800-1323.

Finally, if you have a favorite short program, piece of code, or interesting tip or trick, send it to folklore@plaid.sun.com.

The C program code is shown on the following pages.

```
/*
 * inetgr - interface to inetgr() yp routine to determine if a given
 * machine or user is in a netgroup
 *
 * Daniel Steinberg
 */

#include <stdio.h>

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

{
    char *prog;
    char *netgroup;
    char *mach = NULL;
    char *user = NULL;
    char *domain = NULL;
    char *key, *val;
    int keylen, vallen;
    char thisdomain[256];
    int err;

    prog = argv[0];
    if (argc < 3) {
        fprintf(stderr,
            "usage: %s netgroup machine [user [domain]]\n", prog);
        exit(1);
    }
    if (getdomainname(thisdomain, sizeof(thisdomain)) < 0) {
        fprintf(stderr, "%s: could not get current domain\n", prog);
        exit(1);
    }
    netgroup = argv[1];
    key = netgroup;
    keylen = strlen(key);
    if (yp_match(thisdomain, "netgroup", key, keylen, &val, &vallen)) {
        fprintf(stderr, "%s: no such netgroup as '%s' in %s\n",
            prog, key, thisdomain);
        exit(1);
    }
    if (*argv[2] != '\0') {
        mach = argv[2];
        key = mach;
        keylen = strlen(key);
    }
    if (yp_match(thisdomain, "hosts.byname",
        key, keylen, &val, &vallen)) {
        fprintf(stderr,
            "%s: no such machine as '%s' in %s\n",
            prog, key, thisdomain);
        exit(1);
    }
}
```

```

    }
}
if ((argc > 3) && (*argv[3] != '\0')) {
    user = argv[3];
    key = user;
    keylen = strlen(key);
    if (yp_match(thisdomain, "passwd.byname",
                key, keylen, &val, &vallen)) {
        fprintf(stderr,
            "%s: no such user as '%s' in %s\n",
            prog, key, thisdomain);
        exit(1);
    }
}
if ((argc > 4) && (*argv[4] != '\0')) {
    domain = argv[4];
    key = domain;
    keylen = strlen(key);
    if (yp_match(thisdomain, "networks.byname",
                key, keylen, &val, &vallen)) {
        fprintf(stderr,
            "%s: no such domain as '%s' in %s\n",
            prog, key, thisdomain);
        exit(1);
    }
}
printf("%s %s %s %s %s\n",
    netgroup,
    (err = inetgr(netgroup, mach, user, domain)) ?
    "contains" : "does not contain",
    mach ? mach : "",
    user ? user : "",
    domain ? domain : "");

exit(err ? 0 : 1);
}

```



THE HACKERS' CORNER

THE HACKERS' CORNER 1031

Porting SunView 1031



THE HACKERS' CORNER

Porting SunView

Porting Applications to SunView

This month's **Hackers' Corner** contains three example programs that illustrate processing event-driven input when running SunView.

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.

Input Processing in SunView

Many customers have difficulty when porting an existing program to Sun workstations running SunView. This difficulty is typically in the area of input processing.

To understand the problem, it is helpful to understand how two different programming styles, 'mainline' and 'event-driven', effect where flow of control resides within a program.

Mainline Input Processing

Mainline input processing is the traditional type of flow of control. The flow of control resides within the main program, and the program blocks when it expects input. For example, a C programmer may use `scanf()` or `getchar()` to wait for characters on `stdin` (standard input).

Event-Driven Input Processing

SunView supports event-driven input processing. The program specifies event handlers at initialization time, e.g. via the `WIN_EVENT_PROC` attribute. After initialization, the program passes the flow of control to the notifier with `window_main_loop(base_frame)`.

The notifier calls the specified event handler each time the specified event occurs. After processing the event, the handler returns control to the notifier. The notifier normally returns the flow of control to the main program only when the `base_frame` is destroyed.

The Problem

The *SunView Programmer's Guide*, part number 800-1345, fully describes how to program in the event-driven style, but it does not give many clues about programming in the mainline style. Furthermore, many customers new to SunView may not recognize that the manual is describing something which is incompatible with what they have in mind.

The input handling problem usually occurs when trying to port an existing program to SunView since most existing programs are written in the mainline style. The event-driven style is certainly very well suited to writing window-based applications, with their many different input objects (keyboard, mouse, panel items, pop-up menus, and so forth). But it is often impractical to convert an existing program from mainline to event-driven style.

A Solution

The first step toward a solution is to understand the two programming styles, and to recognize each when you encounter it. New programs should be written in the event-driven style if at all possible, and existing mainline programs will usually have to remain mainline.

Next we need to find a way to perform mainline input in SunView. The manual tells you in the notifier chapter how to use `notify_dispatch()` and `notify_do_dispatch()` to write a mainline program which reads `stdin` and the like, but this does not help when trying to read event-driven input like mouse events and other SunView events. Example programs 2 and 3 below show how this can be done.

The Notifier: What It Is and What It Does

The notifier is *not* as is sometimes imagined some central process which distributes all of the events to the appropriate processes. It is, rather, a set of functions linked in from the `suntool` and `sunwindow` libraries. The notifier is actually a part of your code and runs as your process when you call it.

The notifier's purpose is to collect all of the events directed to your process, and to call the appropriate event handlers for each event. Some of these event handlers will be written by you (setup with `WIN_EVENT_PROC`, or `notify_interpose_destroy_func()`, for two examples). Other event handlers come straight from the libraries (Open, Close, Move, resize the frame, scroll, repaint canvas, and so forth).

Note it is very important to invoke the notifier as frequently as possible, otherwise your application will appear to be 'dead'. If the notifier is not running, it cannot receive requests to `Move` or `Close` the frame, for example. Usually the notifier is running while you are waiting for input. If you become CPU-bound, you should call the notifier from time to time during your computation.

The Notifier: How to Run It

`window_main_loop()` is used by event-driven programs in the normal SunView style.

`notify_dispatch()` runs the notifier once as `notify_dispatch()` processes the first event in each queue. Note that `notify_dispatch()` should be called more than once to be more sure of processing all events. If any

events are pending, it calls the handlers for those events, and then returns. In mainline-style programs, it is useful for keeping the window alive during compute-intensive periods, for catching non-blocking input, and for taking action after calls like `window_set(window, WIN_SHOW, TRUE, 0)`.

`notify_start()` runs the notifier continuously. It returns only if `notify_stop()` is called in an event handler, or if the frame is quit. It can be used to do blocking input.

`notify_do_dispatch()` and `notify_no_dispatch()` turn implicit dispatching on and off. Alternative versions of `read(2)` and `select(2)` are loaded from the `sunwindow` library which run the notifier while blocked when implicit dispatching is on. In this way, your window will not become dead while using `getchar()` and the like.

Please note one warning: if a `notify_stop()` is executed while a function is blocked, it will unblock unsatisfied.

Doing Standard I/O

The `stdio` of a window program is (like any other program) inherited from the environment from which it was invoked. In most cases this means that the `stdio` appears in the `shelltool` or `cmdtool` from which it was run. Please note that input to your window is seen by you as events, while input to the parent `shelltool` is seen by you as `stdin`, and can be read with `getchar()`.

You will need to decide where you want the keyboard dialogue to take place. Many programs do not use any `stdio`, but make use of input events and `pw_text()`. Other programs have `stdio` scattered throughout their code (`printf` and `scanf`, `getchar` and `putchar`, or FORTRAN `READ` and `WRITE`) or require the dialogue to take place on a scrolling, terminal-like area. These programs will need to use `stdio`, and can make use of either the parent `shelltool`, or use a `ttysw` for systems running SunOS release 3.4.

Please note that these programs require SunOS release 3.4 due to the use of `TTY_ARGV_DO_NOT_FORK`. However, SunOS release 3.4 is not required if all one wants to do is use a `STDIO_TTYSW`. Refer to the the example 1 code that follows.

Quitting the Frame

Existing programs are unlikely to understand the 'quit' option in the frame menu. When the quit option is taken, the frame will be destroyed, but the program will *not* automatically exit. In fact, it may not even be aware that the frame has disappeared.

There are three ways to respond in this case.

□ *disable the option*

(Set the Quit item to MENU_INACTIVE.) This method ensures that the problem does not arise, and is used in example program 2.

□ *detect the quit, and cause an exit*

This is a very severe action, much like typing control-C, and the application may consider this undesirable. This method is used in example program 1.

□ *detect the quit, and return status and/or clear a flag*

This is cleanest method, but it may be inconvenient for the application to have to keep testing the status or flag. This method is used in example program 3.

Please note that all examples have a quit handler for the sake of completeness, though in example 2 this can never be called, because the quit option has been made inactive.

Three Example Programs

Three example programs are shown at the end of this article. Example 1 is a simple program using graphic output, but no event-driven input. This is as described in 'Porting Programs to SunView', section 16.6 of the *SunView Programmer's Guide*, part number 800-1345.

Example 2 does event-driven input, using `notify_start()` to block, waiting for input. When the event occurs, an event handler is called which stores the event, and calls `notify_stop()`. This causes the notifier to exit, and so `notify_start()` unblocks. The mainline can then retrieve and process the event.

Example 3 additionally shows such features as non-blocking input, and redirection of ASCII events to the `stdin` window.

Examples 1 and 3 use the symbol `STDIO_TTYSW` to select between `stdio` appearing in the parent shell (0), and appearing in a subwindow of its own (1) for systems running SunOS release 3.4.

All examples are compiled using the command shown below.

```
machine% cc -o example example.c -lsuntool -lsunwindow -lpixrect
```

References for Further Information

The following sections of the *SunView Programmer's Guide*, part number 800-1345, are suggested for details on the topics shown below.

Section	Topic
2.4	Notifier introduction
4	How to create windows
5	Canvases
6	How to interpret the input event
7	The graphic primitives
16	The Notifier
16.6	Explicit and implicit dispatching

In addition to chapter 7 shown above, also refer to the *Pixrect Reference Manual*, part number 800-1254, for `pixrect` details.

Example 1

The code for mainline input example 1 follows. This is the simplest example. It uses `stdio`, and writes graphics to a canvas, but does not accept any event-driven input such as mouse clicks and the like.

Example 1 uses implicit dispatching, as described in section 16.6 of the *SunView Programmer's Guide*, part number 800-1345.

When using implicit dispatching, you will need to find out when the frame is 'quit' by the user, in order to know when to terminate your program. To do so, interpose in front of the frame's destroy event handler with `notify_interpose_destroy_func()` so that you can notice when the frame goes away. At this point we call `notify_stop()` to break the read out of a blocking state.

```
#define STDIO_TTYSW 1                /* Changed to 1. --r */

/*
   Mainline input example 1.
*/

#include <stdio.h>                   /* added. --r */
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/tty.h>

static Frame          base_frame;
static Canvas        canvas;
static Tty           ttysw;
static Pixwin        *pw;

static Notify_value my_notice_destroy();

main()
{
    int s, tty_fd;

    base_frame = window_create(0, FRAME, 0);
    #if STDIO_TTYSW
        ttysw = window_create(base_frame, TTY,
                               WIN_ROWS, 8,
                               TTY_ARGV, TTY_ARGV_DO_NOT_FORK,
                               0);
                               /* deleted line w/ signal. --r */
        tty_fd = (int)window_get(ttysw, TTY_TTY_FD);
        dup2(tty_fd, 0);
        dup2(tty_fd, 1);
    #endif
    canvas = window_create(base_frame, CANVAS, 0);
    pw = canvas_pixwin(canvas);

    notify_interpose_destroy_func(base_frame, my_notice_destroy);
    window_set(base_frame, WIN_SHOW, TRUE, 0);
    notify_dispatch(); /* make the windows appear*/

    notify_do_dispatch();

    while( 1 ) {
        printf("Enter size of square (suggest 100): ");
        scanf("%d", &s);
        if(feof(stdin)) /* added. --r */
            break; /* added. --r */
        if( s == 0 )
            break;
        pw_writebackground(pw, 0, 0,
                           (int)window_get(canvas, CANVAS_WIDTH),
```

```
        (int>window_get(canvas, CANVAS_HEIGHT),
        PIX_CLR);
    pw_writebackground(pw, 10, 10, s, s, PIX_SET);
}
fprintf(stderr, "exiting nicely\n");
exit(0);
}

static Notify_value
my_notice_destroy(frame, status)
    Frame frame;
    Destroy_status status;
{
    if(status != DESTROY_CHECKING) {
        fprintf(stderr, "exiting abruptly\n");
        notify_stop();
        exit(0);
    }
    return( notify_next_destroy_func(frame, status) );
}
```

Mainline Input Example 2

The code for mainline input example 2 follows. This example shows event-driven input. Important features are described below.

Event Input

Event-driven input processing uses `MS_LEFT` and `MS_RIGHT` to write a string to the location of the event on the canvas, and ASCII events are echoed to the canvas.

This is mainline blocking event-driven input, and is an example for traditional CAD/CAE-style programs.

Quit

Finally, note that as an alternative to interposing on our destroy proc as in example 1, here we disable the 'quit' option from the frame's menu.


```

/*
   Mainline input example 2.
*/

#include <suntool/sunview.h>
#include <suntool/canvas.h>

static Frame          base_frame;
static Canvas         canvas;
static Pixwin        *pw;

static Notify_value  getevent_notice_destroy();
static void          getevent_canvas_event_proc();

/* ----- */
main()
{
    Event  *event;
    Window window;

    base_frame = window_create(0, FRAME, 0);
    canvas = window_create(base_frame, CANVAS,
        WIN_CONSUME_KBD_EVENT, WIN_ASCII_EVENTS,
        WIN_EVENT_PROC,      getevent_canvas_event_proc,
        0);
    pw = canvas_pixwin(canvas);

    /* Inactivate the frame menu's "Quit" option. */
    menu_set(
        (Menu)menu_find(
            (Menu>window_get(base_frame, WIN_MENU),
                MENU_STRING, "Quit", 0),
            MENU_INACTIVE, TRUE, 0);

    window_set(base_frame, WIN_SHOW, TRUE, 0);
    notify_dispatch();

    while( get_event(&event, &window )) {
        if (event_is_up(event))
            continue;
        switch(event_id(event)) {
        case MS_RIGHT:
            pw_text(pw, event_x(event), event_y(event),
                PIX_SRC, 0, "Clunk");
            break;
        case MS_LEFT:
            pw_text(pw, event_x(event), event_y(event),
                PIX_SRC, 0, "Click");
            break;
        default:
            if( event_is_ascii(event) )

```

```
        pw_char(pw, event_x(event), event_y(event),
                PIX_SRC, 0, event_id(event));
    }
    break;
}
}
printf("exiting nicely\n");
exit(0);
}

static Event  getevent_event;
static Window getevent_window;

/* ----- */
get_event(event, window)
Event **event;
Window *window;
{
    notify_start();          /* this blocks until notify_stop() */
    *event = &getevent_event;
    *window = getevent_window;
    return ( getevent_window != NULL );
}

/* ----- */
static void
getevent_canvas_event_proc(canvas, event)
Canvas canvas;
Event *event;
{
    getevent_event = *event;
    getevent_window = canvas;
    notify_stop();
}
}
```

Mainline Input Example 3

The code for mainline input example 3 follows. This example shows event-driven input. The major features are described below.

Event Input MS_LEFT writes a string to the location of the event on the canvas, and ASCII events are echoed to the canvas.

This is mainline blocking event-driven input, and is another example for traditional CAD/CAE-style programs.

Compute-Intensive MS_MIDDLE selects a compute-intensive task. `notify_dispatch()` is used to keep the frame and scrollbars alive, and to allow non-blocking input to occur. Input is tested, and, if present, breaks out of the compute loop.

`stdio` The 'new string' option on the menu (selected with MS_RIGHT) reads `stdin` with `scanf()`. This option uses implicit dispatching as described in 'Porting Programs to SunView', section 16.6, of the *SunView Programmer's Guide*, part number 800-1345.

It sets up `WIN_INPUT_DESIGNEE` to redirect ASCII events to the window handling `stdio`, to avoid moving the mouse into that window for `stdin` entry. However, note that ASCII events are consumed when `get_event()` is called.

Scrollbars The canvas uses scroll bars to display a part of a larger bitmap.

Menus Events must be translated from canvas to window space.

Quit A flag is cleared and can be tested when the 'quit' option is selected on the frame menu.

```
#define STDIO_TTYSW 1                /* added. --r */

/*
   Mainline input example 3.
*/

#include <stdio.h>
        /* deleted #include <signal.h>. --r */
#include <suntool/sunview.h>
#include <suntool/scrollbar.h>
#include <suntool/canvas.h>
#include <suntool/tty.h>

static Frame      base_frame;
static Canvas     canvas;
static Tty        ttysw;
static Pixwin     *pw;

/* Support for get_event();          */

static Event      getevent_event;
static Window     getevent_window;
static int        getevent_gotevent;
static int        getevent_continue_flag = 1;
static int        getevent_blocking;
static int        getevent_nonblocking;

/* ----- */
main(argc, argv)
int argc;
char **argv;
{
    Event      *event;
    Window     window;
    Menu       menu;
    static char string[96] = "Click";
    char       str[20];
    int a, i, j, k, y;

    menu = menu_create(MENU_STRINGS,
        "One",
        "Two",
        "Three",
        "New string",
        0,
        0);

    init_windows();

    while( get_event_test_continue() ) {
        get_event(&event, &window);
    }
}
```

```

if (event_is_up(event))
    continue;
switch(event_id(event)) {
case MS_LEFT:
    /*
    show that we can see the x & y position of the event
    */
    pw_text(pw, event_x(event), event_y(event),
            PIX_SRC, 0, string);
    break;

case MS_RIGHT:
    /*
    show that we can use menus
    */
    canvas_window_event(canvas, event);
    notify_no_dispatch();
    a = (int)menu_show(menu, canvas, event, 0);
    notify_do_dispatch();
    canvas_event(canvas, event);
    switch(a) {
    case 1:
    case 2:
    case 3:
        sprintf(str, "%d", a);
        pw_text(pw, event_x(event), event_y(event),
                PIX_SRC, 0, str);
        break;
    case 4:
        printf("Enter a string: ");
        scanf("%s", string);
        break;
    }
    break;
case MS_MIDDLE:
    /*
    Do something compute intensive, use non-blocking
    input to test for interruptions.
    */
    pw_text(pw, 5, 20, PIX_SRC, 0,
            "This will take a while...");
    pw_text(pw, 5, 40, PIX_SRC, 0,
            "Hit a mouse button to interrupt");
    y = 50;
    pw_vector(pw, 0, y, 500, y, PIX_CLR, 1);
    get_event_noblocking(1);
    for(i=0; i<500 ;i++) {
        notify_dispatch();
        if( get_event_test_event() )
            break;
        pw_vector(pw, i, y, i, y, PIX_SET, 1);
        for(k=0; k<2000; k++);
    }
}

```

```

        get_event_noblocking(0);
        pw_writebackground(pw, 0, 0, 400, 45, PIX_CLR);
        break;
    default:
        if( event_is_ascii(event) )
            pw_char(pw, event_x(event), event_y(event),
                PIX_SRC, 0, event_id(event));
        break;
    }
}
fprintf(stderr, "exiting nicely\n");
}

/* ----- */
static Notify_value
getevent_notice_destroy(frame, status)
    Frame frame;
    Destroy_status status;
{
    if(status != DESTROY_CHECKING) {
        getevent_window = (Window)0;
        getevent_continue_flag = 0;
        notify_stop();
    }
    return( notify_next_destroy_func(frame, status) );
}

/* ----- */
static void
getevent_canvas_event_proc(canvas, event)
    Canvas canvas;
    Event *event;
{
    if(getevent_blocking) {
        /* return any event */
        getevent_event = *event;
        getevent_window = canvas;
        getevent_gotevent = 1;
        notify_stop();
    } else
    if(getevent_nonblocking) {
        /* return only 'major' events */
        if( event_is_down(event) )
            switch( event_id(event) ) {
                case MS_LEFT:
                case MS_MIDDLE:
                case MS_RIGHT:
                    getevent_event = *event;
                    getevent_window = canvas;
                    getevent_gotevent = 1;
                    break;
            }
    }
}

```

```

}

/* ----- */
int
get_event_test_event()
{
    return getevent_gotevent;
}

/* ----- */
int
get_event_test_continue()
{
    return getevent_continue_flag;
}

/* ----- */
int
get_event_noblocking(s)
    int s;
{
    getevent_nonblocking = s;
}

/* ----- */
int
get_event(event, window)
    Event **event;
    Window *window;
{
    if( getevent_gotevent == 0 ) {
        getevent_blocking = 1;
        window_set(canvas, WIN_CONSUME_KBD_EVENT, WIN_ASCII_EVENTS, 0);
        notify_start();          /* this blocks until notify_stop() */
        window_set(canvas, WIN_IGNORE_KBD_EVENT, WIN_ASCII_EVENTS, 0);
        getevent_blocking = 0;
    }
    *event = &getevent_event;
    *window = getevent_window;
    getevent_gotevent = 0;
    return ( getevent_window != (Window)0 );
}

/* ----- */
init_windows()
{
    base_frame =
    window_create(0, FRAME,
        FRAME_LABEL, "Mainline input demo",
        0);

    canvas =
    window_create(base_frame, CANVAS,

```

```
    WIN_ROWS,          16,
    CANVAS_WIDTH,      1200,
    CANVAS_HEIGHT,     1200,
    WIN_VERTICAL_SCROLLBAR, scrollbar_create(0), /* changed --r */
    WIN_HORIZONTAL_SCROLLBAR, scrollbar_create(0), /* changed --r */
    CANVAS_AUTO_EXPAND, FALSE,
    CANVAS_AUTO_SHRINK, FALSE,
    WIN_EVENT_PROC,    getevent_canvas_event_proc,
    0);
pw = canvas_pixwin(canvas);

#if STUDIO_TTYSW
ttysw = window_create(base_frame, TTY,
    WIN_ROWS, 8,
    TTY_ARGV, TTY_ARGV_DO_NOT_FORK,
    0);
        /* deleted call to signal. --r */
dup2((int>window_get(ttysw, TTY_TTY_FD), 0);
dup2((int>window_get(ttysw, TTY_TTY_FD), 1);
window_set(canvas,
    WIN_INPUT_DESIGNEE, (int>window_get(ttysw, WIN_DEVICE_NUMBER),
    0);
#else
window_set(canvas,
    WIN_INPUT_DESIGNEE, win_nametonumber(getenv("WINDOW_ME")),
    0);
#endif
window_fit_height(base_frame);

notify_interpose_destroy_func(base_frame, getevent_notice_destroy);
window_set(base_frame, WIN_SHOW, TRUE, 0);
notify_dispatch();
notify_do_dispatch();
}
```

CUMULATIVE INDEX: 1987

CUMULATIVE INDEX: 1987 1049



CUMULATIVE INDEX: 1987



Index

Special Characters

- .cshrc
 - at usage, 211
 - slow, 67
 - with interactive shell, 68
- .login, 67
- /dev
 - ownership, 54
- /etc/group
 - searches, 26
 - YP master server, 27
- /etc/hosts
 - INR, 51
- /tmp
 - with NFS partitions, 355

1

- 1-800-USA-4-SUN
 - device driver calls, 787
 - use of, 763, 975

3

- 3270 SNA
 - bugs, 922

8

- 800 USA-4-SUN
 - use of, 364

A

- ACCELL
 - databases, 271
- accelerator
 - floating point, 700
- adb
 - finding nd servers, 983
- address
 - device drivers, 195
- address mask, 74
- addresses
 - classes of, 391
 - Internet, 391
- alias
 - used with history, 781
- aliases
 - mail, 291
 - namestripes, 220

aliases, continued

- sendmail, 269
- AnswerLine, 5, 26, 67, 219, 291, 401, 605, 737, 797, 1023
 - device driver calls, 787
 - use of, 763, 975
- answermail
 - script, 321, 324
 - script installation, 321
- applications
 - SunView porting, 1031
- architecture
 - Prism, 287
 - Sun4, 403
- ARP, 393
- arrow keys
 - mapping, 265
- asm
 - errata, 689
 - with C source, 215
- assembler
 - bugs, 416
- assembler bugs
 - compilers, 806
- assembly code
 - with C source, 215
- at
 - used with .cshrc, 211
- at (1)
 - answermail script, 321

B

- back-to-back packets, 245
- backups
 - Hackers' Corner, 801
- beta sites, 673, 683
 - questionnaire, 687
- bind
 - port numbers, 213
- blocking
 - using select (), 62
- boot
 - from PROM monitor, 1007
- booting
 - specific kernel, 771
- Bourne shell
 - bugs, 478
- Bourne shell bugs

Bourne shell bugs, *continued*
bugs, 875

bridge box, 719

broadcasting
subnets, 391

brouchure
Sun Education, 688

Browser
installation, 612
program, 611

bsc3270
bugs, 507, 915

bscrje
bugs, 507, 916

buffer
Ethernet, 245

buffers
color frame, 276
frame, 358

bug
3/50 CPU board, 189
reporting, 206

bugs
3270 SNA bugs, 922
assembler, 83, 416, 806
Bourne shell, 140, 478, 875
bsc3270, 103, 507
bsc3270 bugs, 915
bscrje, 103, 507
bscrje bugs, 916
C compiler, 84, 417, 807
C shell, 140, 478, 875
cgi, 120, 454, 853
cgp, 854
compiler general, 835
compiler library, 99, 437, 826
compiler utilities, 102, 444, 836
compilers, 83, 416, 806
dai bugs, 919
Datacomm, 103, 507, 915
debugger, 90, 423
debugger documentation, 839
debuggers, 813, 814
demo, 123
diagnostics, 109, 445, 838
dna, 105, 511
dna bugs, 920
documentation, 111, 447, 839
driver, 460, 857
editor utility, 495, 901
fixed in SunOS 3.5, 988
formatter, 156, 495
formatter utility, 901
FORTRAN compiler, 93, 426
FORTRAN documentation, 447, 840
general utility, 906
gp, 123, 456
graphics, 120, 454, 853
index entries, 347
installation, 490, 898
kernel, 128, 460, 857
kernel general bugs, 859
library utility, 904

bugs, *continued*

linker, 440
lint, 100, 440, 829
LISP, 170, 528
Local 3270 bugs, 920
mail, 158, 497
mail utility, 904
make, 158, 498
make utility, 905
Modula 2, 171, 529
Modula2, 937
network, 135, 470, 868
network general, 474, 871
network library, 135, 470, 868
network program, 137, 474, 872
network protocol, 873
network Yellow Pages, 874
nfs, 135, 470, 868
nse, 938
optimizer, 100, 441, 830
osi bugs, 920
PC-NFS, 530
pixrect, 123, 457, 855
printer, 159, 498
printer utility, 907
program utilities documentation, 841
program utility, 499, 907
protocol, 138, 475
RPC, 874
setup, 490, 899
shell, 140, 478, 875
sna3270, 108, 515
Sun Common Lisp, 936
SunAlis, 167, 504, 912
SunAlis database, 504, 912
SunAlis documentation, 504, 912
SunAlis general, 504, 913
SunAlis spreadsheet, 505, 913
SunCORE, 124, 855
SunCORE documentation, 447, 841
SunCORE graphics, 457
SunGKS, 520, 926
SunGKS library, 520, 926
SunINGRES, 168, 523, 929
SunINGRES documentation, 523, 929
SunINGRES general, 525
SunINGRES general bugs, 932
SunINGRES library, 525, 932
SunINGRES program, 527, 934
SunSimplify, 532, 944
SunSimplify library, 532
SunSimplify program, 532
SunUNIFY, 172, 534, 946
SunView, 142, 480, 879
SunView documentation, 448, 842
SunView general, 890
SunView library, 480, 879
SunView program, 486, 890
SunWindows, 488, 896
syscall, 467, 865
system administration documentation, 844
system administration, 148, 490, 898
system administration documentation, 448
system administration utilities, 494, 899

bugs, *continued*
 transcript, 531, 940
 User documentation, 849
 user manuals, 451
 utilities, 156, 495, 901
 utility programs, 160
 uu`cp`, 165, 501, 909
 vt100 emulation bugs, 924
 vt100`tool`, 108, 517
 windows documentation, 839
 X.25, 517
 X.25 bugs, 925
 yellow pages, 139

Bulletin Board, 250
 bulletin board
 Sun Education, 688

C

C
 calling NeWS, 407
 passing FORTRAN variables, 1013

C compiler
 bugs, 417

C compiler bugs
 compilers, 807

C shell
 bugs, 478

C shell bugs
 bugs, 875

canvas
 colormaps, 282

carrier sense, 724

CDB
 errata, 296

cgi
 bugs, 454

cgi bugs
 graphics, 853

checksum
 Ethernet, 380

child processes
 dbx`tool`, 192
 PID, 192

chip
 83586, 188

client
 sample programs, 13
 stream socket, 12

collisions
 detection of, 724

color, 275
 maps, 276

colormaps, 357

common variables
 passed to C, 1013

compatibility
 Sun4 binary, 403
 SunView 2, 597

compiler
 FORTRAN 1.0 extensions, 707

compiler utility
 bugs, 444

compilers, 806
 assembler bugs, 83, 416, 806
 bugs, 83, 416, 806
 C compiler bugs, 84, 417, 807
 compiler library bugs, 99, 437
 debugger bugs, 90, 423, 813, 814
 FORTRAN compiler bugs, 93, 426
 general bugs, 835
 library bugs, 826
 linker bugs, 440
 lint bugs, 100, 440, 829
 optimizer bugs, 100, 441, 830
 utility bugs, 102, 444, 836

configuration
 custom kernels, 1023

configurations
 controllers, 580
 disks, 580
 Sun-2, 582
 Sun-3, 581

console messages
 program, 743

CONSULT-HSPEED
 high-speed disciplines, 789

CONSULT-`PLOCK`
 lock process text, 789

consulting
 device drivers, 787
 specials, 787

controller
 Ethernet, 245

controllers
 combinations with disks, 581, 582
 disk configurations, 580
 SunOS installation, 775

conventions
 naming common variables, 1015

conversion
 color to monochrome, 358
 hex-to-decimal, 984

corrections
 April TOM, 224
 routing, 296

courses
 device drivers, 792
 Sun Education, 688

CPU
 multiple, 244

CR LF
 end-of-line, 44

CSD Consulting
 device drivers, 787
 specials, 787

cumulative index
 use of, 347

Customer Software Services, 5, 39
customer-training@sun.com
 Sun Education, 688

D

- dai
 - bugs, 919
- DARPA, 73
- database bugs
 - SunAlis database, 912
- databases
 - incompatible, 271
 - SunAlis database bugs, 504
- datacomm, 915
 - 3270 SNA bugs, 922
- Datacomm
 - b3c3270 bugs, 103, 507, 915
 - b3crje bugs, 103, 507, 916
 - bugs, 103, 507, 915
- datacomm
 - dai bugs, 919
- Datacomm
 - dna bugs, 105, 511, 920
- datacomm
 - Local 3270 bugs, 920
 - osi bugs, 920
- Datacomm
 - sna3270 bugs, 108, 515
- datacomm
 - vt100 emulation bugs, 924
- Datacomm
 - vt100tool bugs, 108, 517
 - X.25 bugs, 517, 925
- datagrams
 - fragmentation of, 393
 - reassembly of, 393
- daylight savings time
 - kernel, 24
- dbxtool
 - child processes, 192
- dd (1)
 - slow disk test, 263
- debugger bugs
 - compilers, 813, 814
- Debugger documentation
 - bugs, 839
- debuggers
 - bugs, 423
- Debuggers
 - documentation bugs, 839
- defaults
 - monitor types, 402
- defaultsed
 - mouse, 606
- demultiplexing
 - TCP/IP, 377
- device drivers
 - Consulting Services, 194
 - courses, 792
 - device addresses, 195
 - phone support, 787
 - references, 793
 - third party, 789
- device names
 - SunOS installation, 775
- devices
 - ones present, 301
- diagnostics, 838
 - bugs, 109, 445, 838
- disk
 - combinations with controllers, 581, 582
 - determining configurations, 580
 - enlarging procedure, 569
 - enlarging SunIPC, 569
 - slow test, 263
- disk space
 - saving, 355
- disks
 - size using mkfs, 267
 - size using setup, 267
- dispatching
 - procedures, 567
- DMA, 194
- dna
 - bugs, 511, 920
- documentation, 839
 - bugs, 111, 447, 839
 - debugger documentation bugs, 839
 - FORTTRAN documentation bugs, 447, 840
 - program utilities documentation bugs, 841
 - SunAlis documentation bugs, 504
 - SunCORE documentation bugs, 447, 841
 - SunINGRES documentation bugs, 523
 - SunView documentation bugs, 448, 842
 - system administration documentation bugs, 844
 - User documentation bugs, 849
 - user manual bugs, 451
 - windows documentation bugs, 839
- documentation bugs
 - SunAlis documentation, 912
 - SunINGRES, 929
- DoD, 73
 - critical path specification, 1003
- domain system
 - Internet, 387
- driver
 - bugs, 460, 857
- drivers
 - courses, 792
 - references, 793
 - third party, 789
- DST, 24
 - Australia, 24
 - Europe, 24
 - rules table, 25
- dump
 - nd1 partitions, 266
 - with host names, 270
- dumping tapes
 - Hackers' Corner, 801
- DVMA, 194

E

- editor utility
 - bugs, 495, 901
- education
 - courses, 688

education, *continued*
 SunOS courses, 777
 Educational Services
 courses, 688
 email
 Sun Education, 688
 end-of-line
 definitions, 44
 environment
 answermail variables, 322
 errata, 563, 980
 April TOM, 224
 a sm usage, 689
 disk controllers, 767
 May CDB, 296
 routing, 296
 errno
 EWOULDBLOCK, 64
 errors
 le0, 21
 Ethernet, 380
 back-to-back packets, 245
 buffer, 245
 controller, 245
 header, 380
 throughput, 246, 719
 Europe
 hotlines, 562
 event-driven input, 1031
 Hackers' Corner, 1031
 experiment
 answermail script, 321
 calling NeWS from C, 407
 devices present, 301
 extensions
 FORTRAN 1.0 compiler, 707

F

fast mode
 Weitek chips, 702
 file systems
 maximum size, 1009
 files
 after power failures, 783
 filesystems
 backups, 801
 fixes
 SunOS 3.5, 988
 floating point
 accellerator, 700
 fork ()
 child processes, 192
 formatter
 bugs, 495
 formatter utility
 bugs, 901
 FORTRAN
 1.0 announcement, 707
 1.0 extensions, 707
 compiler bugs, 426
 FORTRAN documentation
 bugs, 447, 840

fragmentation
 datagrams, 393
 frame buffers
 with screendump, 358
 frames
 quitting, 1033
 fsck(8), 985
 ftime, 24
 FTP, 370

G

gateway, 74
 gateways, 390
 general bugs
 compilers, 835
 SunView, 890
 general utility
 bugs, 906
 generic kernels
 configuring, 1023
 getpagesize (), 300
 gettimeofday, 24
 gettytab
 modem entries, 209
 GMT, 24
 gp
 bugs, 456
 gp bugs
 graphics, 854
 graphics, 853
 bugs, 120, 454, 853
 cgi bugs, 120, 454, 853
 demo bugs, 123
 gp bugs, 123, 456, 854
 pixrect bugs, 123, 457, 855
 SunCORE bugs, 124, 457, 855
 grpck
 YP map problems, 27

H

Hackers' Corner
 answermail script, 321
 devices present, 301
 memory size, 299
 NeWS, 407
 SunView, 1031
 survey, 239
 hardware
 color frame buffers, 276
 Hayes-Compatible, 219
 headers
 IP, 379
 octets, 375
 overview, 377
 hexadecimal
 conversion to decimal, 984
 history
 use of, 781
 host names
 with dump, 270
 with fdump, 270

hostid(1)
 tip, 798
 hotline
 Europe, 562
 procedures, 567
 UK, 562
 use of, 363
hotline@sun.COM
 reporting bugs, 206
 hotlines
 world, 976, 979

I

I/O
 sockets, 9
 ICMP, 386
 id, 720
 ie0 spurious interrupt
 SunOS 3.2, 187
 images
 converting to monochrome, 358
 incompatibility
 databases, 271
 index
 bug entries, 347
 inline, 689
 input
 event-driven, 1031
 INR, 51
 requirements for, 53
 installation
 bugs, 490
 SunOS, 775
 installation bugs
 system administration, 898
 Intercon
 hotline, 979
 Internet
 addresses, 391
 domain system, 387
 protocols, 369
 interprocedure interface, 1014
 IP, 369
 headers, 379

K

kernel, 857
 booting specific, 771
 bugs, 128, 460, 857
 configuration, 1023
 daylight savings time, 24
 driver bugs, 460, 857
 general bugs, 462, 859
 swap space, 232
 syscall bugs, 467, 865
 time zones, 23
 keys
 mapping, 265

L

labels
 pedestal, 580
 LANCE, 21
 packets, 21
 layering
 mail, 375
 le0
 errors, 21
 level 1
 network hardware, 732
 level 2
 network hardware, 732
 library bugs
 compilers, 437, 826
 network, 470
 network library, 868
 SunGKS, 926
 SunINGRES, 525, 932
 SunSimplify, 532
 SunView, 480, 879
 library utility
 bugs, 904
 line speeds
 uucp, 214
 linker
 bugs, 440
 lint
 bugs, 440
 lint bugs
 compilers, 829
 LISP
 bugs, 170, 528
 Lisp
 quick check, 698
 Local 3270
 bugs, 920
 localtime, 25
 lockd
 needing statd, 589
 long variables
 word boundaries, 1018
 lost+found
 missing, 985
 script to restore, 985, 986
 lpr
 flow control, 43

M

mail, 371
 aliases, 291
 bugs, 497
 formats, 293
 layering, 375
 pitfalls, 293
 routing, 389
 systems, 353
 transport systems, 354
 user agents, 353
 Mail Service, 250
 mail utility
 bugs, 904

make
 bugs, 498
 make bugs
 nse, 938
 make utility
 bugs, 905
 management
 SunTrac software, 1000
 manuals
 proprietary, 197
 maps
 color, 276
 YP, 34
 mask
 address, 74
 maxusers
 setting, 1024
 memory
 size, 299
 SunAlis requirements, 251
 SunINGRES requirements, 259
 mkfs
 disk sizes, 267
mkfs(8), 985
 MMU, 366
 modems
 gettytab entries, 209
 high speed, 693
 software installation, 693
 Modula 2
 bugs, 171, 529
 Modula2, 937
 bugs, 937
 monitors
 defaults, 402
 determining type, 401
 high-resolution, 358
 mouse
 defaultsedit, 606
 MS-DOS, 569

N

namestripes
 aliases, 220
 reprogramming, 27
 naming convention
 read, 243
 transfer, 243
 write, 243
 naming conventions
 common variables, 1015
 ND
 swap space, 229
 nd servers
 using adb, 983
 using ypmatch, 984
 nd1
 dumping partitions, 266
 network, 868
 bugs, 135, 470, 868
 general bugs, 474, 871

network, *continued*
 library bugs, 135, 470, 868
 nfs bugs, 135, 470, 868
 program bugs, 137, 474, 872
 protocol bugs, 138, 475, 873
 RPC bugs, 874
 yellow pages bugs, 139, 874
 networks
 carrier sense, 724
 collision detection, 724
 Ethernet theory, 723
 hardware problems, 731
 loopback, 738
 performance of, 727
 Q & A, 733
 services, 737
 thin Ethernet, 731
 newfs
 dumping partitions, 266
 NeWS
 called from C, 407
 with SunView 2, 595
 NFS, 372
 bugs, 470
 partitions, 57
 partitions with root, 355
 nfs bugs
 network, 868
 nodes
 multiple, 244
 notifier
 definition of, 1032
 running, 1032
 SunPro", 938
 bugs, 938
 make bugs, 938

O

octets
 TCP/IP headers, 375
 optimizer, 689
 bugs, 441
 optimizer bugs
 compilers, 830
 osi
 bugs, 920
 out-of-band data
 sockets, 9

P

packets, 380
 back-to-back, 245
 LANCE, 21
 page faults
 overview, 366
 panic: iechkcca, 187
 partition
 calculating size, 230
 swap space, 229
 partitions
 dumping nd1, 266
 read protection, 57

passing
 FORTRAN variables, 1013
 PC-NFS
 bugs, 530
 pedestal
 information, 580
 Personal AnswerLine, 5
 PERT analysis
 SunTrac, 1001
 PF keys
 mapping, 265
 physmem, 299
 PID
 child processes, 192
 ping, 577
 script, 578
 pixrect
 bugs, 457
 pixrect bugs
 graphics, 855
 pmap_rmtcall, 578
 port number
 assignment of, 213
 porting
 SunView, 1031
 PostScript
 pscat output, 198
 setlinewidth, 208
 pounds sterling
 symbol printing, 49
 power failures
 diskless workstations, 783
 printer
 bugs, 498
 printer utility
 bugs, 907
 printing
 images, 357
 Prism
 windows, 287
 procedure
 enlarging SunIPC disk, 569
 hotline, 567
 products
 release levels, 349, 560, 681, 766, 978
 program
 FORTRAN to C, 1013
 program bugs
 network, 474, 872
 SunINGRES, 527, 934
 SunSimplify, 532
 SunView, 486, 890
 Program utilities documentation
 bugs, 841
 program utility
 bugs, 499, 907
 PROM monitor
 using boot, 1007
 proprietary manuals, 197
 protocol
 bugs, 475

protocol bugs
 network, 873
 pscat
 PostScript, 198
 ptroff
 pounds sterling, 49
 pty
 ownership, 54

Q

questionnaire
 beta sites, 687
 quitting
 frames, 1033
 quota
 delays, 605
 symbolic links, 606

R

rdump
 with host names, 270
 read
 naming convention, 243
 read optimization, 59
 reduced time, 591
 write permission, 59
 read protection
 NFS, 57
Read This First
 purpose, 584
 reassembly
 datagrams, 393
 recomputation
 floating point, 700
 recovery
 RPC timeouts, 773
 references
 device drivers, 793
 register
 saving D2, 46
 release level
 SunOS, 205
 releases
 software products, 349, 560, 681, 766, 978
 SunOS 3.5, 987
 reporting bugs, 206
 RETRN
 end-of-line, 44
 root
 access, 355
 file permissions, 57
 read permissions, 57
 rotdelay, 591
 routing
 mail, 389
 RPC, 577
 timeout recovery, 773
 timeouts, 773
 RPC bugs
 network, 874
 rpc.etherd, 578

`rpc.rstatd`, 578

RTF

purpose, 584

Rutgers University, 369

S

SCB, 188

SCLISP

quick check, 698

screendump, 357

color windows, 288

screenload, 358

script

answermail, 324

restoring `lost+found`, 986

SCSI

slow disk test, 263

seek

read optimization, 59

select ()

exceptions, 64

non-blocking mode, 62

sendmail, 353

aliases, 269

zero-length messages, 797

server

stream socket, 10

SunView 2, 596

setlinewidth, 208

setup

bugs, 490

disk sizes, 267

setup bugs

system administration, 899

shell, 875

Bourne shell bugs, 140, 478, 875

bugs, 140, 478, 875

C shell bugs, 140, 478, 875

shoebox

disk labels, 581

SIGIO, 9

SIGPIPE

server, 10

SIGQUIT

server, 10

SIGURG, 9

sleep, 43

SMTP

application example, 384

sna3270

bugs, 515

sockets

example programs, 10

out-of-band data, 9, 15

programming examples, 9

servers, 10

well-known, 381

Software Information Services, 1, 39

specials

CSD Consulting, 787

device drivers, 787

specific kernel

booting, 771

spreadsheet bugs

SunAlis spreadsheet, 505, 913

statd

with `lockd`, 589

STB

duplication of, 181

stdio

read optimization, 59

window programs, 1033

subnets

address mask, 74

broadcasting, 391

definition, 73

enabling, 77

Exterior Gateway Protocol, 73

limitations, 75

SunOS release 3.3, 264

subnetting, 73

Sun Common Lisp, 936

bugs, 936

Sun Education

device driver course, 792

SunOS courses, 777

sun!hotline

reporting bugs, 206

use of, 363

sun!stb-editor, 26, 39, 67, 70, 181, 219, 291, 401, 605, 737, 797, 1023

sun!sunbugs

reporting bugs, 206

suncustomer-training

Sun Education, 688

Sun4

architecture, 403

binary compatibility, 403

SunAlis, 912

bugs, 167, 504, 912

database bugs, 504, 912

documentation bugs, 504, 912

general bugs, 504, 913

memory requirements, 251

release 2.0, 249

spreadsheet bugs, 505, 913

support of, 592

windows, 252

SunAlis 1.0

discontinued support of, 592

SunAlis 2.0

upgrade program, 592

sunbugs@sun.COM

reporting bugs, 206

SunCGI, 280

SunCore, 282

SunCORE

bugs, 855

documentation bugs, 447, 841

SunCore

printing images, 357

SunCORE bugs

- SunCORE bugs, *continued*
 - graphics, 855
 - SunCore documentation
 - bugs, 841
 - SunCORE graphics
 - bugs, 457
 - SunGKS, 926
 - bugs, 520, 926
 - library bugs, 520, 926
 - SunGKS library
 - bugs, 520
 - SunINGRES, 929
 - bugs, 168, 523, 929
 - documentation bugs, 523, 929
 - general bugs, 525, 932
 - installing release 5.0, 258
 - library bugs, 525, 932
 - memory requirements, 259
 - program bugs, 527, 934
 - release 5.0, 254
 - SunIngres 5.0
 - upgrade workaround, 1010
 - SunIPC
 - enlarging disk, 569
 - SunLink Internet Router, 51
 - requirements for, 53
 - SunOS
 - determining release of, 205
 - installation, 775
 - network services, 737
 - release 3.3 and subnets, 264
 - release 3.5, 987
 - SunSimplify, 944
 - bugs, 532, 944
 - library bugs, 532
 - program bugs, 532
 - Suntools
 - exiting, 55
 - suntools
 - frame buffers, 277
 - reprogramming namestripes, 27
 - SunTrac
 - release 1.0, 1000
 - SunUNIFY, 946
 - bugs, 172, 534, 946
 - SunUnify
 - diskful configuration, 705
 - diskless configuration, 706
 - installation, 703
 - SunView, 879
 - bugs, 142, 480, 879
 - color frame buffers, 278
 - documentation bugs, 448, 842
 - general bugs, 890
 - Hackers' Corner, 1031
 - library bugs, 480, 879
 - porting applications, 1031
 - program bugs, 486, 890
 - SunWindow bugs, 488, 896
 - SunView 2
 - changes, 597
 - compatibility issues, 597
 - SunView 2, *continued*
 - differences from SunView, 595
 - introduction, 595
 - SunView documentation
 - bugs, 448, 842
 - SunWindow
 - bugs, 488
 - SunWindow bugs
 - SunView, 896
 - super eagle
 - file system size, 1009
 - support
 - discontinuation of, 592
 - swap space
 - ND, 229
 - SunINGRES, 782
 - switcher (1)
 - colormaps, 287
 - symbolic links
 - quota, 606
 - syscall
 - bugs, 467, 865
 - system administration, 898
 - bugs, 148, 490, 898
 - documentation bugs, 448, 844
 - installation bugs, 490, 898
 - setup bugs, 490, 899
 - utility bugs, 494, 899
 - system administration documentation
 - bugs, 844
 - system administration utilities
 - bugs, 494
- ## T
- tables
 - software release levels, 349, 560, 681, 766, 978
 - tape drives
 - SunOS installation, 775
 - tape dump
 - Hackers' Corner, 801
 - tape verification, 210
 - TCP, 369
 - sockets, 12
 - TCP/IP
 - demultiplexing, 377
 - references, 394
 - telnet, 44, 370
 - Bridge terminal server, 44
 - terminal
 - tty problems, 590
 - testing
 - beta sites, 683
 - The Hacker's Corner
 - Browser, 611
 - console messages, 743
 - thin Ethernet
 - specification, 731
 - throughput
 - Ethernet, 246, 719
 - time zones
 - TZ, 23

time zones, *continued*

uucico, 23

timeouts

recovery options, 773
RPC, 773

training

Sun Education, 688

transcript, 940

bugs, 531, 940

transfer

naming convention, 243

troff

previewing output, 198

tty

ownership, 54

virtual, 590

tunefs (8)

read times, 591

TZ, 23

DST rules table, 25

U

UDP, 386

UK

hotline, 562

underscores

appended in C, 1014
trailing in C, 1014

UNIX

monitoring status of, 577

update, 783

upgrade

SunAlis program, 592

USA-4-SUN

use of, 364, 567

USAC

feedback, 365

User documentation

bugs, 849

User Documentaiton

documentation bugs, 849

user manuals

bugs, 451

utilities, 901

bugs, 156, 495, 901

editor bugs, 495, 901

formatter bugs, 156, 495, 901

general bugs, 906

library bugs, 904

mail bugs, 158, 497, 904

make bugs, 158, 498, 905

printer bugs, 159, 498, 907

program bugs, 499, 907

utility program bugs, 160

uucp bugs, 165, 501, 909

yellow pages, 33

utility bugs

compilers, 836

system administration, 899

uucico

time zones, 23

uucp

uucp, *continued*

bugs, 501, 909

Hayes-Compatible, 220

line speeds, 214

V

variables

answermail environment, 322

common, 1013

long word boundaries, 1018

verification

tapes, 210

vi

maps, 69

vt100 emulation

bugs, 924

vt100tool

bugs, 517

W

Weitek

fast mode chips, 702

well-known sockets, 381

windows, 276

color frame buffers, 277

Prism, 287

server-based, 596

with SunAlis, 252

Windows documentation

bugs, 839

word boundaries, 1018

workaround

SunINGRES, 782

SunIngres 5.0, 1010

world hotlines, 979

introduction, 976

write

naming convention, 243

write permission

read, 59

wstat

floating point, 701

X

X.11

with SunView 2, 595

X.25, 61

bugs, 517, 925

Y

yellow pages, 31

installation, 32

mail aliases, 291

utilities list, 33

Yellow Pages bugs

network, 874

YP, 31

clients, 31

domains, 32

installation, 32

maps, 34

master server, 32

YP, *continued*

rpc, 33

server maps, 31

slave servers, 31

utilities list, 33

ypbind, 32

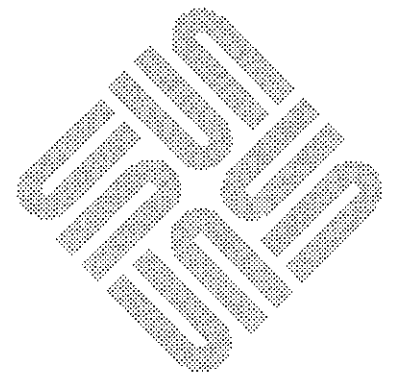
ypmatch

finding nd servers, 984

ypserv, 32

Revision History

<i>Revision</i>	<i>Date</i>	<i>Comments</i>
FINAL	December 1987	Eleventh issue of Software Technical Bulletin (Software Information Services).







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

