sun microsystems

Software Technical Bulletin
December 1987

Software Information Services

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

sun®
microsystems

Software Technical Bulletin
December 1987

Software Information Services

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

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**.

Contents

1

NOTES & COMMENTS

1

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 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 `f77` 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	(011-61-2) 957-2522
	Lionel Singer Group	(011-61-2) 957-2655
Canada	Montreal Branch	(514) 879-1914
	Ottawa	(613) 748-9617
	Vancouver Branch	(604) 641-1296
	Western Branch	(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	(011-81-3) 497-4676
	Nihon Sun	(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

2

ARTICLES

Using adb

Using adb: Determining Your nd Server

You can use adb as in interactive, general-purpose debugger to conveniently to 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 1fe
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: `-lcgi` 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 `-ffpa` 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 `rexd` 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 `stepiing`.

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 `writev(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: `rasfilter8to1.1g` 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 `ndl1`.

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 deamon `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.

```
(number of arcs - number of nodes) / 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	250 250	nodes per diagram arcs per diagram
Trac	4000 4000 200	nodes total per analysis arcs total per analysis 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.

3

STB SHORT SUBJECTS

3

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 `tar`red.

- *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.

4

IN DEPTH

4

IN DEPTH

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.

5

QUESTIONS, ANSWERS, HINTS, AND TIPS

5

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.

```
/*
 * innetgr - interface to innetgr() 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 = innetgr(netgroup, mach, user, domain)) ?
           "contains" : "does not contain",
           mach ? mach : "",
           user ? user : "",
           domain ? domain : "");

    exit(err ? 0 : 1);
}
```

6

THE HACKERS' CORNER

6

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.

Whe 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 STDIO_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();
}
```

7

CUMULATIVE INDEX: 1987

7

CUMULATIVE INDEX: 1987

Index

Revision History

Revision	*Date*	*Comments*
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