IRIS Workstation Software Distribution

Release GL1-W2.1

Silicon Graphics, Inc.
630 Clyde Court
Mtn. View, CA 94043

Document Number 5001-065-001-1

IRIS Workstation Software Distribution Release GL1-W2.1 Document number: 5001-065-001-1

CONTENTS

1. INTRODUCTION

This document describes Release GL1-W2.1 of the IRIS workstation software. Release 2.1 is an update to Release 1.7 IRIS workstation software and includes the UNIX† operating system and IRIS graphics library. In this document "Release 2.1" and "2.1" are used as a short-hand for "Release GL1-W2.1".

Silicon Graphics Inc. (SGI) has found the ethernet hardware performs more reliably when its bus address is set to "0x7ffc". This new address was compiled into the kernel for Release 2.1 on the IRIS 1400. This change is not necessary for the IRIS 1500. *Before booting your IRIS 1400 workstation with the new software, you must change the bus address of the ethernet board.* This involves inserting "jumpers" onto the ethernet board. Instructions to perform this operation appear in Appendix D. If you would rather that SGI perform these steps, call the SGI **Geometry Hotline** telephone number listed below.

There are four chapters and four appendices in this document.

Chapter 1 contains this introduction.

Chapter 2 contains information on how to load and install the 2.1 software from the distribution tape.

Chapter 3 describes the differences between Releases 1.7 and 2.1.

Chapter 4 contains general information.

Appendix A contains updated information on system administration.

Appendix B contains information on stand-alone disk formating.

Appendix C contains information on new or significantly changed manual pages.

Appendix D contains information on the Excelan ethernet board address change procedure.

The information included here is intended as a supplement to the *IRIS Workstation Guide*, which includes detailed information about the workstation and UNIX system administration and usage.

---

† UNIX is a trademark of AT&T Bell Labs.

SGI provides a comprehensive product support and maintenance program for the IRIS terminal and workstation. For further information contact the SGI Geometry Hotline:

SGI Geometry Hotline	
(800) 252-0222	North America except California (toll-free)
(800) 345-0222	California (toll-free)
(415) 962-0606	Worldwide (collect)

2. UPDATING TO GL1-W2.1

This chapter provides instructions for reading the new software off the update distribution tape and installing it on workstations running older versions of software. It also includes instructions for installing the update over the network, from a workstation with a tape drive to a workstation that has no tape drive. If you have a workstation that has no tape drive, note that you must update that workstation before applying the update to the workstation with a tape drive. In the event you have more than one machine with no tape drive, you must update all of them before updating the workstation that has a tape drive.

Most new files from the tape directly overwrite the old versions on the disk. There are two categories of files that are not directly overwritten: site-dependent system configuration files, and programs that may be executing as the update occurs. The system administrator will be able to perform appropriate editing of the configuration files; installating executing programs is handled by a script supplied on the tape.

In updating from version W1.7 to version W2.1, it is necessary to recompile all user programs in order to run under the new version of the operating system.

The distribution tape contains the standard system software. Optional software is delivered on separate tapes with appropriate installation instructions.

If a problem arises at any step of the update process, determine and correct the cause of the problem before proceeding. If you are unable to determine the cause of the problem, contact the Geometry Hotline. Failure to successfully perform any step can render the system unusable.

2.1 Preparation

This section provides procedures for backing up your file system and for providing enough space for the new release of software. First back up any systems with no tape drive, then back up the system(s) with tape drive(s). Afterward, the procedure for ensuring that there is enough space for the update (Section 2.1.3) should be followed for each system that will be updated.

2.1.1 Backing Up a System with No Tape Drive

This procedure requires that the system with no tape drive, referred to here as system "B", can access another workstation over the network, called system

"A", and that system A has a tape drive installed.

1. System A must be in multi-user mode.

2. Make sure there are no other users on system B.

3. Reboot system B and leave it in single user mode:

```
/etc/reboot -q
```

4. The standard configuration files are designed to perform a file system check automatically when the system comes up in single-user mode. On system B, if the configuration files have been altered, or if for some reason the file system check is not done, initiate one manually:

```
/etc/fsck
```

   Correct any errors that are found.

5. Bring system B to multi-user mode and log in as root.

6. Back up the system. You can back up the entire system on one tape if it will fit; otherwise, back up the root file system on one tape and the **/usr** file system on another. Three procedures follow. The first backs up the entire system on a tape. This is followed by two procedures: backing up only the root file system and backing up only the /usr file system.

   Backing Up The Entire System To One Tape

   A. Install a tape cartridge in system A.

   B. On system B, change to the root of the file system.

```
cd /
```

   C. On system B, mount the /usr file system.

```
mount /dev/md0c /usr
```

   or, for model the 1500:

```
/etc/mount /dev/ip0c /usr
```

   Mount any other file systems on the system as in the previous step.

D. On system B, begin the transfer.

```
cpio -0a1 . | xx [name of system A] dd ibs=20b\
 obs=600b of=/dev/rmt1
```

Backing Up Only the Root File System to One Tape

A. On system B, unmount the user file system by typing:

```
/etc/umount /dev/md0c
```

or, for model 1500 users:

```
/etc/umount /dev/ip0c
```

B. If you have any additional mountable file systems they should be unmounted. Change to the root of the file system.

```
cd /
```

C. Install a tape cartridge in system A.

D. On system B, back up the root file system:

```
cpio -0a1 . | xx [name of system A] dd ibs=20b\
 obs=600b of=/dev/rmt1
```

Backing Up Only the /usr File System to One Tape

A. Install a tape cartridge in system A.

B. On system B, mount the user file system:

```
/etc/mount /dev/md0c /usr
```

or, for model 1500 users:

```
/etc/mount /dev/ip0c /usr
```

C. On system B, back up the /usr file system:

```
cd /usr
cpio -0a1 . | xx [name of system A] dd ibs=20b\
 obs=600b of=/dev/rmt1
```

D. If you have any additional mountable file systems they should now be backed up in a similar way.

2.1.2 Backing Up a System with a Tape Drive

1. Make sure there are no other users on the system.
2. Reboot the system to leave it in single user mode:

```
/etc/reboot -q
```

3. The standard configuration files are designed to perform a file system check automatically when the system comes up in single-user mode. If the configuration files have been altered, or if for some reason the file system check is not done, initiate one manually:

```
/etc/fsck
```

   Correct any errors that are found.

4. Bring the system to multi-user mode and log in as root.
5. Back up the entire system. If you do not have a standard procedure for backup, follow this procedure (assuming a cartridge tape drive is installed):

   A. Unmount the user file system by typing:

```
/etc/umount /dev/md0c
```

   or, for model 1500 users:

```
/etc/umount /dev/ip0c
```

   B. If you have any additional mountable file systems they should be unmounted.

   C. Install a tape cartridge and back up the root file system:

```
tar -c /
```

   or:

```
find / -print | cpio -ovh1
```

D. Mount the user file system again:

```
/etc/mount /dev/md0c /usr
```

E. Back up the file system:

```
tar -c /usr
```

or:

```
find /usr -print | cpio -ovh1
```

F. If you have any additional mountable file systems they should now be backed up.

2.1.3 Ensuring There Is Enough Space For The Update

Make sure there is enough space to read in the update tape. This is done by comparing the size of each of the major directories on the tape (given in the table below) with the size of those directories on your system. If the directory on the tape is significantly larger, the increase in size must be available as free space. The totals of the increases for the directories on each file system (root and /usr) will indicate how much tree space should be available on that file system. If you have added files to any of the listed directories, their sizes must be taken into account.

Note that if a directory on the tape is smaller than the directory on the disk, the apparent "extra space" will not be available while reading in the tape. This is because the files on the disk will still exist as the tape is being read.

If the calculations show that more free space is required, existing disk files must be deleted on the appropriate file system (root or **/usr**). (Make sure that you have a good backup of the files you delete.)

Space required	
Directory	*Blocks (kb)*
/bin	2830
/dev	7
/etc	1156
/kernels	690
/lib	405
/stand	141
/usr/adm	10
/usr/bin	4238
/usr/dict	192
/usr/games	1330
/usr/include	465
/usr/lib	3643
/usr/local	5
/usr/mail	1
/usr/man	2035
/usr/news	1
/usr/people	6
/usr/preserve	1
/usr/pub	1
/usr/spool	26

To determine how much space is presently being used on your system the following commands may be issued:

```
du -s     /bin /crash /dev /ate /kernels /lib /stand /usr/adm

du -s     /usr/bin /usr/dict /usr/games /usr/include /usr/lib

du -s     /usr/local /usr/mail /usr/man /usr/news /usr/people

du -s     /usr/preserve /usr/pub /usr/spool
```

2.2 Installation

Installation differs slightly between a workstation with no tape drive and one with a tape drive. It differs in how the data is read onto the disk (the software load), and occurs at the beginning of the installation procedure. The software load procedure differs, but the remaining part of the procedure is identical for systems with and without tape drives. Execute the appropriate software load for the machine at hand, the follow the instructions common to both types of machine in Section 2.2.3. The software installation should be performed on all workstations with no tape drive before it is performed on the workstation with a tape drive. This is because of incompatibilities between the new kernel (version 2.1) and the old kernel (version 1.7).

2.2.1 Software Load On A Workstation With No Tape Drive

1. Make sure that the system is running in the normal mode as root:
   A. The system is multi-user mode.
   B. The user file system, **/usr**, is mounted.
   C. You are logged in as root.
   D. You are in the root directory, /.
2. Make sure system A is in multi-user mode with.
3. Insert the 2.1 update tape into the system A tape drive.
4. Read in the data by typing the following on system B:

   ```
   (xx [name of system A] dd ibs=600b obs=20b if=/dev/rmt1) |
     cpio -iduv
   ```

   It will take 20 minutes or more to read in the tape.
5. Proceed to Section 2.2.3.

2.2.2 Software Load On A Workstation With A Tape Drive

Caution

Be sure to follow the instructions in this section only after updating any workstations that do not have a tape drive.

1. Make sure that the system is running in the normal mode as root:

   A. The system is multi-user mode.

   B. The user file system, **/usr**, is mounted.

   C. You are logged in as root.

   D. You are in the root directory, /.

2. Put the update tape in the tape drive and read it in:

   ```
   cpio -ihduv1
   ```

   It will take 20 minutes or more to read in the tape.

2.2.3 Common Installation Procedure

1. There is a shell script (it was read from the tape) that verifies the existence of certain critical files. These files must be present before the system is rebooted, or the system may not come back up again. Run the script by typing:

   ```
   /verify2.0
   ```

   (This script is valid for release 2.1). If any errors are reported, do not continue with the next step. The problem must be solved before the system can be rebooted. If you cannot solve the problem, seek help by calling Silicon Graphics at the one of the numbers listed on page 2 of this manual.

2. If there is an older version of the vmunix kernel in the root directory, move it out of the way

   ```
   mv /vmunix /ovmunix
   mv /vmunix1 /ovmunix1
   ```

3. Install the new kernel:

   ```
   cp /kernels/vmunix /vmunix
   ```

```
cp /kernels/vmunix1 /vmunix1
cp /vmunix /defaultboot
```

4. Take the system down (after making sure that there is no activity such as other users working) by typing the following:

   ```
   /etc/reboot -q
   ```

5. If your system is a 1400 you must now change the Ethernet bus address. Follow the instructions in Appendix D or contact the Hotline.

6. Reboot the system with the new kernel. The method for booting depends on the model of the system you have (the PROMs differ).

   On a 1400 with PROMs version less than "3.0", type the following:

   ```
   d
   ```

   On a 1500 and 1400s with PROMs version "3.0" or greater, type the following:

   ```
   b
   ```

7. The standard distribution configuration files are designed to perform a file system check when the system comes up. If the file system check has not been automatically performed, initiate one manually. Correct any errors that appear:

   ```
   /etc/fsck
   ```

8. There will be a shell script in the root directory that performs various "cleanup" functions after the update tape is read in. Execute it by typing:

   `/update2.0` **model**

   where **model** is either 1400 or 1500. (This script is valid for release 2.1). This script will install the programs that have been protected from being overwritten, "ranlib" the new libraries, delete some old extraneous versions of programs and libraries from the disk, save the old versions of configuration files, and make sure the proper devices are present. This script will normally inform you of errors, or that it has completed successfully.

9. Update the configurations files. Following is a list of file names that may need to be altered with site-specific information.

/.cshrc	/.login
/.logout	/.profile
/etc/TZ	/etc/bcheckrc
/etc/brc	/etc/checklist
/etc/cshrc	/etc/gettydefs
/etc/group	/etc/inittab
/etc/motd	/etc/passwd
/etc/profile	/etc/rc
/etc/sys_id	/etc/termcap
/etc/ttytype	
/usr/lib/Mail.help.~	/usr/lib/Mail.help
/usr/lib/Mail.rc	/usr/lib/acct/holidays
/usr/lib/crontab	
/usr/lib/uucp/L-devices	/usr/lib/uucp/L-dialcodes
/usr/lib/uucp/L.cmds	/usr/lib/uucp/L.sys
/usr/lib/uucp/USERFILE	

Configuration files that were left in standard distribution form under the previous release can be left in standard form in the new release as well. Changes that were made in the configuration files under the previous release should now be made in the new versions.

The old versions of the configuration files were saved during the execution of the "update2.0" script by appending a dot to the name, before moving the new version into place. Thus for each of these files, a side-by-side comparison can be made to indicate what changes are necessary. See *diff (1)*.

10. Verify that critical files are present:

```
/verify2.0
```

If any errors appear, make sure you correct them before proceeding.

11. Reboot the system again to ensure proper operation. On a 1400 with PROMs version less than "3.0", type the following:

```
/etc/reboot -q
d
```

On a 1500 and 1400s with PROMs version "3.0" or greater, type the following:

```
/etc/reboot -q
b
```

12. Bring the system up multi-user.

13. Verify that the system is functioning correctly.

14. The files in the above list are renamed when the new versions of the files are loaded. The older files have a "." appended to the name. Once the system is up, clean up by removing the old configuration files whose names end with ".".

15. At this point you can restore any files that were deleted to make space during the preparation for installing the update.

16. Recompile all user object modules, libraries and programs.

17. There is a script on the tape that checks for "duplicate" executable files that exist, under the same name, in more than one directory in the search path. This indicates that an old version remains, and needs to be removed. For each "duplicate" found, the script will produce an "ls –l" listing, showing the age of the file. In general, the older of a set of two or more files should be removed, but the system administrator should verify this for each file the script finds. To run the script, type:

    ```
    /dup2.0
    ```

    while in multi-user mode. (This script is valid for release 2.1).

3. OVERVIEW

This chapter contains general information about this release, including a comparison of Release 1.7 to Release 2.1 and known deficiencies in Release 2.1.

3.1 Software Products

This release is composed of the following software products:

Product	Status
UNIX System V with virtual memory	Included
XNS Ethernet communications & serial comm.	Included
IRIS "C" Graphics Library	Included
IRIS FORTRAN Graphics Library	Included
IRIS Remote "C" Graphics Library	Included
IRIS Remote FORTRAN Graphics Library	Included
IP/TCP Ethernet communications	Optional
Fortran Compiler	Optional
Pascal Compiler	Optional
IRIS Terminal Programming Environment	Optional
Reconfigurable Kernel	Optional

The IRIS workstation can act as a host to IRIS terminals. The two machines communicate using one of the communications facilities. Refer to the *IRIS Workstation Guide* for more information.

Using the IRIS Terminal Programming Environment, you can reprogram the standard graphics library. This allows you to add routines to the terminal program that can be called by the host. You can also add interactive code segments that run locally, eliminating network and remote host delays.

The Reconfigurable Kernel allows you to add new device drivers to UNIX.

3.2 Demos

The directory **/usr/people/demos** contains programs that demonstrate some of the many capabilities of the IRIS workstation. The programs also demonstrate

some uses of the IRIS workstation; some are actually used by SGI customers in their work.

There is documentation on the demos in Appendix C of this release notice and also in Section 1 of the the on-line manual.

3.3 Gifts

The directory /usr/people/gifts contains the sources to unsupported programs from SGI. Many of these programs include useful techniques that can save you the trouble of re-inventing them. The following sources are supplied:

1. 4.2.bit.c — Bit-wise operations for FORTRAN.
2. color.c — Compose colors by varying the amounts of the three primary colors: red, green, and blue. Once the desired color has been created, the C or FORTRAN code to generate it can be displayed.
3. colors.c — Similar to **color** but up to eight colors can be displayed simultaneously for comparison.
4. cube.c — A program, similar to **planets**, that can display, rotate, translate, and scale a cube.
5. fan.c — Display a fan of many colors.
6. gcmd.c — Issue some simple graphics commands such as **cmov**, **charstr**, **clkoff**, etc.
7. gplanes.c — Display the number of available bitplanes.
8. hyper.c — A program, similar to **planets**, that can display, rotate, translate, and scale *n*-dimensional cubes (hypercubes).
9. ndcube.f — A FORTRAN program, similar to **hyper.c**, that can display, rotate, translate, and scale *n*-dimensional cubes (hypercubes).
10. planets.c — The **planets** program that displays a solar system in motion, complete with moons and a space ship.
11. planets.f — The FORTRAN version of the **planets** program.
12. quad.c — Display four ellipses.
13. setdate.c — Allows you to set the system date with the mouse.
14. sqiral.c — Display a square spiral made up of 500 segments that demonstrates the speed of the workstation in displaying complex objects.

15. track.c — Display a square that tracks the cursor as the mouse is moved.
16. vms.bit.c — More bit-wise operations for FORTRAN.

3.4 Special Files

This section provides a list of the special files in **/dev** and a short description of each file.

1. console — The console terminal (IRIS).
2. floppy — The optional floppy disk drive.
3. drum — Paging device (used for virtual memory).
4. kmem — Kernel memory.
5. md0[a-h] — Disk 0 file systems (1400 only).
6. md1[a-h] — Optional disk 1 file systems (1400 only).
7. ip0[a-h] — Disk 0 file systems (1500 only).
8. ip1[a-h] — Optional disk 1 file systems (1500 only).
9. mem — Memory.
10. mt1 — Cartridge magnetic tape.
11. nrtape — Cartridge magnetic tape (no rewind on open or close).
12. null — The null device (zero length on input, data sink on output).
13. pty? — Master pseudo tty devices.
14. rmd0[a-h] — Disk 0 file systems raw devices (1400 only).
15. rmd1[a-h] — Optional disk 1 file systems raw devices (1400 only).
16. rip0[a-h] — Disk 0 file systems raw devices (1500 only).
17. rip1[a-h] — Optional disk 1 file systems raw devices (1500 only).
18. rmt1 — Cartridge magnetic tape (treated as a blocked device).
19. rqic — Cartridge magnetic tape (treated as a blocked device).
20. rmt2 — Cartridge magnetic tape, no rewind on **open** or **close** (treated as a blocked device).

21. swap — Swap device.
22. syscon — System console (usually linked to console).
23. systty — System console (usually linked to console).
24. tty — One's own tty device.
25. ttyd[123] — Serial tty ports for terminals, printers, and modems. They can each be directly connected to a modem or through a *null modem* to a terminal or printer.
26. ttym2 — Serial tty port 3 with modem control (peripheral device must supply DSR, or DTR through a *null modem*, for device port to be opened by program). If **ttym2** is used, then **ttyd2** must not be used and vice versa.
27. ttyn* — XNS ethernet ports.
28. ttyp* — Slave pseudo ttys.

3.5 Changes in Configuration Files

This section describes the changes made to system configuration files since the Release 1.7. The descriptions are grouped by directory.

/ (the root directory)

.cshrc — Aliases for **multi** and **single** have been removed, since there are shell scripts to do these functions.

/etc

termcap — The color used for the cursor and underlined text (for **man**) has been changed from blue to green.

/usr/lib/uucp — The files in this directory were shipped empty. Examples of the contents of these files are given here; change them to suit your system needs.

L-devices It now has an owner and group of *uucpadm*. In Release 1.7 it was owned by *root*.

Example contents:

```
DIR         ttyd1   ttyd1   4800
DIR         ttyd2   ttyd2   4800
DIR         ttyd3   ttyd3   1200
DIR         ttyd3   ttyd3   300
DIR         xns     xns     xns
```

L-dialcodes It now has an owner and group of *uucpadm*. In Release 1.7 it was owned by *root*.

Example contents:

```
ba 415
bost 617
boston 617
nv 212
sf 415
sj 408
wash 202
```

L.cmds It now has an owner and group of *uucpadm*. In Release 1.7 it was owned by *root*.

Example contents:

```
rmail
rnews
date
finger
lpr
ls
ps
who
```

L.sys It now has an owner and group of *uucpadm*. In Release 1.7 it was owned by *root*.

Example contents:

```
#The following is 1 line in the file:
gendir1 Any ttyd1 4800 ttyd1 "" \r\c ogin:--ogin:-EOT-ogin:
-BREAK3-ogin: uucpco assword: secret

#The following is 1 line in the file:
genphone Any ttyd3 1200 ttyd3 "" \r\d\r\c S-q\r\d\r\c-S-q
\r\d\r\c-$-q\r\d\r\c-$ K\c DIAL: 5\d5\d5\d1\d2
\d1\d2\d\r\c ONLINE! \r\c ogin:-\r\c-ogin:
-\r\c-ogin:--ogin:-EOT-ogin:-BREAK3-ogin: uucp assword: censored
```

USERFILE — It now has an owner and group of *uucpadm*. In Release 1.7 it was owned by *root*.

Example contents:

```
uucp,       /usr/spool/uucp  /usr/spool/uucppublic
uucpco,     /
,           /usr/spool/uucp  /usr/spool/uucppublic
```

3.6 Major Changes to UNIX Software Since Release 1.7

3.6.1 Virtual Memory

Virtual memory has been implemented. Programs may be as large as 14Mb (swap space permitting). The implementation is such that the sticky bit (see *chmod*(2)) is no longer significant; also the operating system turns it off in some circumstances.

3.6.2 New Object Format

The 2.1 release of the IRIS workstation introduces a new object format that is similar to 4.2bsd systems. This new object format includes a new set of language tools that run significantly faster than the equivalent tools for UNIX System III (which were used in previous IRIS workstation releases). Several user-visible changes to the system occurred between Release 1.7 and Release 2.1 due to this change of object format:

1. Although non-graphics programs that executed under Release 1.7 will execute normally under Release 2.1, recompilation will result in a slightly smaller and usually faster program due to the difference in the way the stack is grown (stack probes are no longer necessary due to virtual memory).

2. Object files (**.o** files) and archives (**.a** files) are not compatible between 1.7 and 2.1. All object files must be recompiled. Object module archives must be recompiled and re-archived before they will be usable in the 2.1 environment.

3. Certain switches to the loader have changed. If you run the loader, *ld*(1), directly, consult the man page supplied in this release in Appendix C and in the on-line manual.

3.6.3 DBX Symbolic Debugger

Release 2.1 contains an initial release of the symbolic debugger, **dbx**(1), for use with C programs. To use this debugger, programs must be compiled and loaded with debugging information (see *cc*(1)). Consult the *dbx*(1) man page in Appendix C for more information.

3.6.4 FORTRAN Graphics Parameter Type Corrections

There are a number of inconsistencies between the documentation and code in the FORTRAN version of the graphics library in Release 1.7. These inconsistencies are divided into two classes.

The first class of inconsistencies is due to changes made to increase efficiency. These were in the routines **clipli**, **clippn**, **screen**, **lampon**, **lampof**, **setbel**, and **keyboa**. These routines have now been made consistent between the environments and are as documented in the *IRIS User's Guide*.

The second class of inconsistencies is errors in the documentation. These errors affect all FORTRAN graphics libraries — remote, native, terminal, and workstation. These errors were in the parameter declarations for the routines **defras**, **deftex**, and **delete**, and were corrected in the second edition of Pipeline. They are reproduced below:

1. In the routine **defras** IRIS User's Guide, (p 2-85, p 3-31), the FORTRAN declaration of the **chars** and **raster** arrays should be:

```
character*8 chars(nc)
character*(*) raster
```

2. In the routine **deftex** IRIS User's Guide, (p 2-80, p 3-32), the FORTRAN declaration of the **tex** array should be changed to:

```
character*(*) tex
```

3. In the routine **delete** IRIS User's Guide, (p 2-54, p 3-33), the parameter count should be declared

   in C as:

```
long count;
```

   in FORTRAN as:

```
integer count
```

in Pascal as:

```
count:longint;
```

4. The **subroutine** declaration was omitted from the FORTRAN declaration of the routine **getdep** IRIS User's Guide, (p 2-114, p 3-51).

3.6.5 The Remote Graphics Libraries

Workstation users can now run graphics on an IRIS terminal from their workstations using the remote graphics library. The remote graphics libraries for both C and FORTRAN are provided in this release. These environments are easily loaded by simply using the *-Zr* switch with the *f77*(1) or *cc*(1) command. Note that C programs must be both *compiled* and *loaded* using this switch. FORTRAN binary (**.j**) files are usable in either environment.

3.6.6 Dealing with Systems that Do Not Boot

Many customers unintentionally make changes to UNIX configuration files or other files critical to UNIX operation, which prevent the UNIX Software from booting. Steps have been taken to remedy this problem.

First, the init program has been enhanced so that the UNIX Software will still be able to boot if some of the entries in the critical files are wrong or the files are missing. This will yield a usable system that can be used to repair the problem. This facility can be used by persons having some familiarity with UNIX Software or under the direction of the SGI Hotline.

Second, some other techniques for recovery are documented. These are discussed in detail in Section 4.7.

3.7 Bug Fixes and Enhancements Since Release 1.7

Release 2.1 provides the following bug fixes and enhancements.

3.7.1 UNIX Software

1. Running graphics from ttyd1 no longer locks up ttyd2.
2. **Tar** includes the following enhancements:
    a. The **-a** flag has been added to prevent **tar** from altering the access time of files read for backing up.
    b. Some techniques have been documented for using **tar** across the ethernet between systems. Some problems with these techniques and how to deal with them have also been documented.
3. **Init** now accepts the new run-state supplied in response to its prompt when the single-user shell is exited or if the **inittab** file is missing. This run-state determines whether to enter multi-user or single-user mode. Previously **init** would not accept whatever answer was supplied under these circumstances, preventing UNIX Software from functioning.
4. Programs no longer hang if they try to malloc() more memory than the system physically has. Program size is limited to the smaller of 14Mb or the size of the swap area (typically less than 8Mb).
5. **Xcp** now **chowns** files so that they will be owned by the invoker of xcp.
6. The disk copy of the date is now updated when **reboot** is used to reboot the system.
7. The C compiler now generates better bit-field code.
8. **Xcp**'s reliability has been improved.
9. **Xcp** now gives diagnostics on failure.
10. **Malloc()** has been replaced with a much faster version.
11. **Cc -o foo foo.c** now deletes **foo.o**.
12. Running **xx** without arguments no longer causes a core dump.
13. **Cc** no longer strips off the eighth bit on literal strings causing `"\200"` to be treated as `"\0"`.
14. The **gtty()** and **stty()** system calls, which were provided for UNIX Version 7 compatibility, are no longer supported.
15. The **cp** program now closes files on error.
16. The **cat** program no longer prints two error messages on write errors.

17. The **echo**, **head**, **ls**, and **tail** programs now detect write errors.
18. **Library** now makes **.j** files, instead of **.obj** files.
19. **Factor** now completely factors all numbers.
20. A typographical error in the *eqn*(1) manual page has been corrected.
21. The *uuto*(1) manual page entry now reads *User* instead of *Logname*.
22. The **adb** program now allows access to variables with long names.
23. **Cc** now compiles **usr.bin/acct/acctcom.c** (the bug related to a class of floating-point expressions).
24. The *ls*(1) manual page entry now documents **-T**.
25. The *cu*(1) manual page entry now documents how to send **BREAK**s from **cu**.
26. The **/etc/rc** file has been corrected to use **/bin/hostname** instead of **/etc/hostname** when invoking **sgboot** and **sgbounce**.
27. The **calendar** program now recognizes days of the week, a certain date every month, and adjusts for holidays. See *calendar*(1).
28. The **tset** sequence used in **.login** in Release 1.7 conflicted with 4.2bsd. It has been changed in Release 2.1.
29. There were both aliases and scripts for **single** and **multi**. The aliases have been removed.
30. **/usr/include/pwd.h** now declares the functions, allowing compatibility with 4.2bsd.
31. The **/usr/include/utmp.h** file now declares the functions, allowing compatibility with 4.2bsd.
32. The dates in **/usr/lib/acct/holidays**, now also used by **calendar**, are now based on January First being 001, as required.
33. The **lpr** program no longer fails if umask is 077.
34. The **lp** programs now work.
35. Invoking a graphics program on the console no longer hangs ttyd3.
36. **Cc** now knows how to compile **.f** files.
37. As per UNIX System V specifications, reading past EOF (through EOF) on a disk device now returns a count of zero (like a plain file), rather than returning an error, as happens in Version 7.

38. The **cc** manual pages are now on-line.

39. Flow control on the 1400 is fixed; previously IXOFF did not work.

40. **Xlogin**'s shell escape no longer disables interrupts. This means that you can now interrupt subshells and the csh history works when using **xlogin** `~!` escape.

41. The `-depth` option for **find** is now documented. The UNIX System V.2 **find** capabilities have been ported and documented.

42. The **Mail** program now properly parses headers for remote mail. Also, you can now "save" a message to a shell command such as **lpr** via:

```
s <message numbers> |<shell command>
```

43. The manual section *getut*(3c) is now correct in regard to the declaration for **pututline**().

44. **Stty** no longer prints `cread` instead of `-cread` if `cread` is disabled.

45. The 4.2bsd **/bin/test** program has been ported. All the remaining features of the System V Bourne shell's builtin **test** have been added.

46. The files **L-devices**, **L-dialcodes**, **L.sys**, **L.cmds**, and **USERFILE**, in **/usr/lib/uucp** and **/usr/spool/uucp/LOGFILE** are now supplied and correct (they were missing or in error in Release 1.7).

47. The sequence:

```
rcs -u foo
```

    no longer causes a core dump.

48. If **sdiff** is given an input line longer than 256 characters it will no longer loop infinitely.

49. One serial port can now be enabled for modem (DTR) control.

50. The header file **/usr/include/sgimath.h** now correctly declares the long float lceil().

51. Termcap now uses green instead of blue for the cursor and underline sequence.

52. **Smt** works.

53. **Csh** no longer causes a core dump when the following is entered:

```
% cd !#:h
```

54. The **passwd** program can now be used to change the password of an account name longer than eight characters, without explicitly specifying the account name. Release 2.1 also corrects a race condition that in Release 1.7 occurred during simultaneous invocation by different users and resulted in corruption of the password file.

55. **Rm** now also accepts 4.2bsd-style flags. Additionally, an argument consisting solely of a "–" will terminate flag processing to allow deletion of files whose first character is a dash.

56. The **-b** flag to **grep** and **egrep** now works.

57. The **copyin** and **copyout** routines now have a 4095 byte limit.

58. This release allows you to make multiple-tape backups using either **tar** or **cpio**.

3.7.2 Graphics

1. **Blink()** is implemented.

2. It is unwise to run two graphics programs concurrently. In this release, a second graphics program starting up will be killed by the system.

3. A warning will be issued if RGB mode is attempted with less than 24 bitplanes.

4. You can use **tpblank** to prevent UNIX Software from changing the color map or otherwise affecting the console IRIS screen when a graphics program exits. UNIX Software never affects colormap entries above seven. **Tpon**(1) may be used to allow UNIX Software to subsequently affect the screen for non-graphics programs.

5. The program **reshape**(1) has been provided to shape the text port.

3.8 Known Problems

1. Terminals turned off or disconnected from enabled ports cause spurious interrupts. This is a hardware problem that is being addressed.

2. You cannot **transform()** more than 300 points with one command.

3. UNIX System V shared memory features are not implemented.

4. Some include files in **/usr/include/sys** are in violation of System V specifications and may cause some application programs not to compile or run correctly.

5. The `who -r` command reports the incorrect run-level when the system is in single-user mode.

6. **Dbx** has several bugs. See *dbx*(1) in Appendix C.

7. XON protocol printers may not work. This is being investigated.

8. The 3D portion of the **Curve** demo does not work.

9. **smt fsf x**, where x is a number, does not work. A workaround is to type multiple "smt fsf" lines.

10. Because of a problem in this release, printer output is limited to 300 baud. See Section 4.5.2.

11. This release contains **uucp** administration files that are zero length. They need to be configured before **uucp** will work.

Other bugs in the system should be reported to the Geometry Hotline. In all cases a bug report should include the smallest possible test case that exhibits the bug, the corresponding source, and an exact sequence of events that cause the failure. Where appropriate, a cartridge tape or a half-inch 1600 bpi tape containing the source and data should be supplied.

3.9 Miscellaneous

3.9.1 Determining the Software Release Version Number

The version of the software release is recorded in the file **/version**. The file contains ascii text.

4. REFERENCE INFORMATION

This chapter contains updated general information about the use of the IRIS workstation. This information will appear in a future revision of the *IRIS Workstation Guide*.

4.1 Booting with new PROMs

The PROMs in an IRIS 1500 contain PROMs that are level 3 or higher. IRIS 1400s shipped after late January 1985 also have level 3 PROMs. These PROMS differ from IRIS 1400 PROMs that are lower in number than level 3. The most notable differences are how the IRIS is "booted" and configuration switch settings. These are described in Appendix A.

4.2 IRIS Workstation as Terminal Host

The IRIS workstation can be used as a "boot server" to service terminals or workstations over the Ethernet. For instance, you can boot an IRIS terminal by reading the **iris** program off the workstation. This is a use of the workstation as a boot server. The boot server and the **iris** program are included as a standard part of Release 2.1. The boot server consists of daemon programs called **/etc/sgboot** and **/etc/sgbounce**. They can be started from **/etc/rc** when the system enters multi-user mode during booting. The different versions of the **iris** program (for different communications media and IRIS terminal type) are in the **/usr/local/boot** directory.

Also, a remote version of the graphics library is supplied which enables a program running on the workstation to display graphics on an IRIS terminal.

4.3 IRIS Terminal Programming Environment

The IRIS Terminal Programming Environment is software that allows you to customize the terminal program that is downloaded to an IRIS terminal. It includes the tools and source files necessary to create and modify the terminal program on an IRIS workstation. This terminal program can then be downloaded into the terminal using the workstation as a host. It may also be used to create stand-alone programs that may be booted from the terminal's floppy. This optional software package is documented in *The IRIS Terminal Programming*

Environment (document number 5001-021-001-0).

4.4 The IRIS Terminal Emulator

This is an optional software package that allows you to use an IRIS workstation as an IRIS terminal. Its operation is described in *wsiris*(1) in Appendix C.

4.5 Serial Ports

This section describes how to use the workstation's serial ports tor terminals, printers, modems, etc.

4.5.1 Port 2 DTR

On some older IRIS 1400s, DTR (Data Transmit Ready) is not supplied on port 2 (ttyd1). An SGI Engineering Change Order resolves this. Customers requiring this change should contact the SGI Geometry Hotline.

This problem can also be resolved by jumpering pin 6 to pin 20 on the cable connecting the workstation to the serial device.

4.5.2 Connecting CRTs and Printers to the Workstation

There are three serial ports on the IRIS 1400 and 1500 workstations (in addition to the console). They are designed to connect directly to DCE (Data Computer Equipment) such as modems, via a straight cable. A *straight cable* has pin 1 of the connector on one side connected to pin 1 of the other connector, pin 2 to pin 2, etc:

Straight cable		
Workstation	*Modem*	*Signals*
1	1	Chassis ground
2	2	Transmit data
3	3	Receive data
4	4	Request to send
5	5	Clear to send
8	8	Carrier detect
6	6	Data set ready
22	22	Ring indicator
20	20	Data terminal ready
7	7	Signal ground

To connect the workstation to DTE (Data Terminal Equipment) such as CRTs and printers, a different cable arrangement, known as a *null modem*, must be used. The following table lists the cabling of a null modem. The pin numbers separated by commas (,) should be connected together and also to the other pin or pins listed in the same row:

Null Modem		
Workstation	*Terminal*	*Signals*
1	1	Chassis ground
2	3	Transmit data
3	2	Receive data
4,5	8	Request to send, Clear to send
8	4,5	Carrier detect
6,22	20	Data set ready
20	6,22	Data terminal ready
7	7	Signal ground

For instance, pins 6 and 22 should be connected on the workstation, and their wires should be connected to pin 20 on the connector for the terminal end of the cable.

This will work with any serial device that complies with the IEEE RS232C specification (except for the problem discussed in Section 4.5.1). However, most printers will work with the following simpler cabling (which assumes that modem control is not used):

Simplified Null Modem		
Workstation	*Terminal*	*Signals*
1	1	Chassis ground
2	3	Transmit data
3	2	Receive data
	4,5	Request to send, Clear to send
	6,8,20	Carrier detect, Data set ready
7	7	Signal ground

Most terminals do not require the various handshaking lines such as *Clear to send* or *Data set ready* and so will work with the following *three-wire null modem*:

Three-wire Null Modem		
Workstation	*Terminal*	*Signals*
2	3	Transmit data
3	2	Receive data
7	7	Signal ground

4.5.3 Hardware and Software Names for Serial Ports

UNIX Software refers to each serial (RS232) port by a name other than that used by the hardware (back panel labeling). The following table documents the correlation:

Names of Serial Ports	
Hardware (back panel)	*Software (***/dev** *&* **inittab***)*
Port 1	console
Port 2	ttyd1
Port 3	ttyd2
Port 4	ttyd3

The following table shows which type of *Null Modem,* if any, to use between different types of equipment:

Null Modem Usage		
From	*To*	*Cable*
1400,1500,2400,2500	Modem	Straight cable
1400,1500,2400,2500	1000,1200,2000,2200	Null modem
1400,1500,2400,2500	Printer	Simplified null modem
1400,1500,2400,2500	CRT	Three-wire null modem
1000,1200,2000,2200	Host	Straight cable (usually)

4.5.4 Serial Ports for Non-login Use

When a port is used for logging in, the software changes the ownership and permissions of the device file. The port cannot then be used for other purposes, such as for connecting a printer or for initiating a connection to another computer with **cu** or **uucp**, without first changing these modes.

Before using a port for any non-login purpose, make these changes to its modes (after turning off the **getty**, see Section 4.6):

1. Change the mode to 666.
2. Change the owner to root.
3. Change the group to sys.

For example, to connect a printer to port 4 (which the software refers to as **/dev/ttyd3**), as root issue the following UNIX commands after any editing to inittab and subsequent rebooting:

```
chmod 666 /dev/ttyd3
chown root /dev/ttyd3
chgrp sys /dev/ttyd3
```

4.5.5 Setting the Baud Rate for Printers

UNIX software will automatically set the baud rate and other terminal modes for ports used for logging in and those used for calling out with **cu** and **uucp**. It will not do so for ports that will be used for printers.

Since UNIX Software will automatically reset the baud rate and terminal modes of any serial device after all programs that opened the device file have closed it (or exited), you must first run a program that opens the device file and keeps it open. While any port can be used, the following example uses **/dev/ttyd**3:

```
sleep 10000000 < /dev/ttyd3&
sleep 3
```

The first **sleep** will run in background, keeping the port open for 4 months, functionally forever. The second **sleep** will give the first one time to start up. Once the port is open, an **stty** command must be issued to specify:

1. The required baud rate.
2. Whether or not X-on/X-off protocol is desired.
3. Whether to precede each transmitted line-feed (newline) with a carriage return.
4. Parity.
5. Etc.

A dash (–) in front of an option to **stty** means turn it off; otherwise it will be turned on. See *stty*(1) and *termio*(7). For example, to use a printer at 1200 baud with X-on/X-off protocol, without supplying a carriage return before each line-feed, issue the command:

```
stty 1200 ixon -onlcr opost < /dev/ttyd3
```

The system will set some of these parameters by default. To determine the defaults, issue the command:

```
stty -a < /dev/ttyd3
```

The –a flag causes **stty** to show the values of all parameters.

The necessary commands can be incorporated into the **/etc/rc** file after the *Daemons* section. After the existing line in **/etc/rc** that reads:

```
echo "."
```

you could add:

```
sleep 10000000 < /dev/ttyd3&
sleep 3
stty 1200 iron -onlcr < /dev/ttyd3
```

Then, whenever the system is rebooted and brought into multi-user mode, the printer's port will be set up automatically.

The **lpr** lineprinter spooler requires that the device to be used as the printer be linked to the file **/dev/lp**. Thus, if the printer is connected to port 4 (**/dev-/ttyd3**), issue the commands:

```
rm -f /dev/lp
ln /dev/ttyd3 /dev/lp
```

This command need be issued only once (unless the printer is subsequently moved to a different port). Once this is done, the command:

```
lpr foo
```

will print the file **foo**.

4.6 Modem Control

Normally the serial ports do not have modem control. That is, the device connected to them (modem, terminal, printer, etc.) does not have to supply Data Set Ready (DSR) or Data Terminal Ready (DTR) if the device is connected through a "*null*modem". Port 3, usually referred to by the software as ttyd2, can be configured in software to be sensitive to DSR (or DTR). Modem control is useful if a modem will be connected to a port to allow dialing into the system over the phone. It will allow the workstation to print the login prompt after someone actually dials in.

To create the device, login as root and issue the commands:

```
mknod /dev/ttym2 c 3 130
chmod 666 /dev/ttym2
```

This only has to be done once. To enable logging in over that port, edit **/etc/inittab**, changing all occurrences of **ttyd2** to **ttym2**, and removing any x that may appear between the first pair of colons (:), and then reboot the system with reboot.

If you do not wish to enable logging in on **ttym2**, you must disable **ttyd2** by editing **/etc/inittab**, putting an x between the first pair of colons (:), and then rebooting.

4.7 Recovering Systems that Do Not Boot

Unintentional changes to files critical to UNIX operation can prevent the UNIX operating system from booting. These files are:

Critical Files	
File	*Purpose*
/	Root directory
/vmunix	Kernel
/vmunix1	Kernel that boots disk1
/dev	Device directory
/dev/console	Console tty device
/dev/syscon	System console device
/dev/systty	System tty device
/dev/md0a	Root file system device (1400 only)
/dev/ip0a	Root file system device (1500 only)
/dev/rmd0a	Raw root file system device (1400 only)
/dev/rip0a	Raw root file system device (1500 only)
/dev/swap	Swap device
/dev/drum	Paging device
/etc	Miscellaneous file directory
/etc/init	Program that starts other user processes
/etc/inittab	Table for **init**
/bin	Commands directory
/bin/su	Used to set environment before **exec**ing shell
/bin/sh	Root shell
/bin/csh	Root Shell

This chapter provides three methods of recovery.

First, UNIX software has been enhanced so that it can handle the most common of these errors.

Second, a version of the UNIX kernel is supplied that will boot from the second disk. This is useful to customers who have a second disk with a copy of the files needed to boot UNIX software.

Third, a stand-alone program is supplied that will load a tape into the root file system on the disk even if the data on the disk has been lost. This destroys any data that was on the root file system of the disk.

4.7.1 Recovery with INIT

In addition, a spare copy of **tar** has been installed in **/etc**. **Init** is able to recover from most other errors that would prevent UNIX software from booting and operating. Recovery from most of these errors is automatic in that the no action is required of the operator to bring the UNIX software to single-user mode (or multi-user if you elect to make your system secure, as discussed in Section 4.8). Normally, **Init** does not repair the cause of the problem. Once the system comes up, you should repair the system. This requires that one or

more files be edited, or loaded from a backup tape. It can require that modes (permissions) of certain files, ownership, or group be changed.

NOTE

This recovery procedure uses **tar** to read files from a tape you supply that are missing from or damaged on the disk. You should make this tape shortly after your system arrives and again after you modify any configuration files or install new software. To make this tape, put a blank tape in the tape drive and type the following:

```
tar - cv /vmunix /vmunix1 /bin /dev /etc
```

Another part of the recovery procedure (that uses **/lastditch**) requires that you have a special-format tape with a "shell" program on it (i.e., **sh** or **csh**). This is called the "last ditch" tape. The shell must be copied onto a tape (either before disaster strikes or on a different system) by issuing the command:

```
dd bs=512 < /bin/csh > /dev/rmt1
```

When UNIX software is booted, it starts **init** and **init** starts a single-user shell. It does this by invoking `su -`. This causes the **su** program to start up. **Su** looks in **/etc/passwd** for an account called *root* and starts a shell (of the type specified in the passwd entry; the default is **/bin/sh**).

The system will not boot correctly if any of the following conditions exists:

1. The **/etc/passwd** file is missing.
2. The entry for *root* is missing or in error in the **/etc/passwd** file.
3. Any of several critical programs is missing or corrupt.

The new **init** senses a problem when it is unable to execute **su** or when **su** executes too quickly. "Quickly" is defined as less than 15 seconds the first time any error occurs and then as less than 30 seconds. If **init** cannot execute **su**, **init** prints the error message:

`execlp of` **/bin/su** `failed; errno =` *n*

where *n* is an error number listed in the introduction to Section 2 in the *UNIX Programmer's Manual*. If **su** executed but finished too quickly, the following message is typed:

```
SINGLE USER MODE
SU is broken
```

This may be preceded by an error message supplied by **su** explaining why it executed so quickly.

In either case, **init** tries to execute **/bin/sh**. If **sh** fails in either of the ways discussed above, init prints either the error message:

```
execlp of /bin/sh failed; errno = n
```

or:

```
SINGLE USER MODE
/bin/sh is broken
```

In either case, **init** then tries to execute **/bin/csh** in a similar manner.

If this fails, **init** tries to read the missing files from the tape drive. This requires that you have a tape in the drive that contains a **tar** backup of the root file system:

```
/bin/tar -xv /vmunix /vmunix1 /bin /dev /etc vmunix vmunix1 bin dev etc
```

Init attempts to read in the critical files that are needed for proper operation of UNIX software but are apparently missing or corrupted. Since the files (and directories) are listed both with and without a leading slash, they are read in regardless of whether or not they were saved on tape with a leading slash.

If the correct tape is not in the tape drive, insert the correct tape and wait for **init** to try **tar** again or reboot after inserting the tape. In this case, do not press the reset button while either of the disk access lights (beside the tape drive) is lit.

If this fails (possible because there is not a **tar** tape in the tape drive), then **init** will invoke the command:

```
/etc/tar -xvpU
```

This attempts to read all the files from the tape and will then change the permissions, ownership, and group of the copy on disk to be the same as the version stored on tape, removing any existing copy on disk which may be the wrong type of file. There is an extra copy of **tar** in **/etc**; you normally use the copy in **/bin**. This can be used to read in any files on the system that may be missing, even devices in **/dev**. If necessary, a custom tape can be produced containing just the missing files so that you do not overwrite data such as programs with older versions from tape (which may be a backup tape if you use **tar** for backup).

This exhausts the **init** program's automatic techniques for recovery. It defers to the user to attempt to recover from more difficult problems. This requires UNIX expertise. Please contact the Geometry Hotline if you require assistance. **Init** allows the you to recover by reading in and executing a program from a custom tape (the "last ditch tape") to try to solve the problem. It does this by attempting to open **/dev/rmt1**, **/dev/rqic**, or **/dev/mt1** until it has successfully opened one of them. If it has failed to open any of the three, init will use the **mknod()** system call to create the device **/rmt1** as the tape drive (with major and minor devices of 13/0) and open it. If any of these opens is successful, then **init** will create the file **/lastditch** with mode 755 and will copy data from the tape drive to **/lastditch**. It will then close these files.

Regardless of whether or not **init** is able to open the tape drive device and read its data into **/lastditch**, it issues a **sync()** system call, wait a few seconds, and execute **/lastditch**.

For example, if the **/bin** directory is unusable (perhaps because **/bin** was changed to an ordinary file during a crash, or because of permissions problems), the shell can be read off the "last ditch" tape. To use this tape, insert the tape into the tape drive and reboot the system by pressing the reset button (when the disk access light is off). **Init** reads the shell off of the tape and executes it. When the shell prompt appears, issue the commands:

```
/etc/unlink /bin
/etc/fsck
```

Then follow these steps:

1. Reboot the system (since the tape drive is unusable after loading **csh** into **/lastditch**).

2. Insert a **tar** tape containing the programs in **/bin** and wait for **init** to invoke **tar**.

Alternatively, you can remove the tape after the **fsck**, reboot the system and wait until **init** executes **/lastditch** (remember that if the previous methods fail, **init** will execute **/lastditch** even if it could not open the tape device file). You are now free to use **tar** and could issue the following command to restore **bin**:

```
/etc/tar -xv /bin
```

Note that the backup copy of tar in **/etc** was used. Significant system experience is needed to solve these types of problems.

Due to a firmware limitation, after reading a tape into **/lastditch** (if there was a tape in the drive), the system will not be able to access the tape drive until after a reboot. Thus, if after executing **/lastditch**, you want to read data from a different tape, you must: 1) remove the tape that was read into **/lastditch**, 2) reboot the system, 3) wait for **/lastditch** to execute (even though there was no

tape to read into it this time), and 4) insert the second tape that will be read (probably by some other method such as **tar**, under user control).

If all these attempts fail, **init** creates and uses its own copy of the console device in the file **/console** (as device 0/0) and tries the whole sequence again (having assumed that it failed because **/dev/console** is missing or corrupt).

If all these attempts fail, **init** assumes the problem is a corrupted file system (as opposed to the permissions, ownership, or incorrect data in some files) and issues the following command:

```
/etc/fsck /dev/md0a /dev/md1a
```

Following the **fsck**, the entire sequence of **su**, **/bin/sh**, etc. is started again.

Init initially attempts to open the device file **/dev/syscon** and use it for terminal I/O for the aforementioned boot sequence. Since **/dev/syscon** is just a link to **/dev/systty**, if init can't open **/dev/syscon**, then it will delete it and attempt to re-link it to **/dev/systty**. If this fails, then init will attempt to link both **/dev/systty** and **/dev/syscon** to **/dev/console**. If this fails, then it will attempt to link both **/dev/systty** and **/dev/syscon** to **/dev/ttyd1**, which will be presumed to be connected to an ordinary terminal running at 9600 baud. If this fails, then **init** will create its own version of the console, as discussed previously. **Init** will then try the entire sequence of **su**, **/bin/sh**, etc.

4.7.2 Recovery by Booting from the Second Disk

When a system has two disks and copies of the critical system files are kept on the second disk, you can boot from the second disk if you cannot boot from the first disk. To boot from the second disk, after pressing the reset button (presumably after shutting the system down normally or after a crash) respond to the IRIS prompt:

```
iris>
```

by issuing the command:

```
d md(1,)vmunix1
```

or, for model 1500 users:

```
b ip1a:vmunix1
```

followed by a RETURN. After the system comes up with the superuser shell prompt, run **fsck** immediately on all file systems. To do this, the file systems should be listed explicitly on the command line and the block device should be specified in all "a" file systems, e.g.:

```
fsck /dev/md1a /dev/rmd1c /dev/md0a /dev/rmd0c
```

or, for model 1500 users:

```
fsck /dev/ip1a /dev/rip1c /dev/ip0a /dev/rip0c
```

At this point the previous root file system (**/dev/md0a** or **/dev/ip0a**) can be mounted and the damage that prevents booting from it can be repaired.

When booting off the second disk, the second partition of this disk is used for swap space instead of using **/dev/swap**. You may have used the second partition of the second disk for temporary space (i.e., /tmp). This space will be used for swap space when booting from this disk. If you do this, use **mkfs** to recreate /tmp on this disk after you begin booting off the first disk again.

If you use your second disk for backup (via a **dd** of the first disk), all the critical files will already be on the second disk. If the second disk is not used for backup, a subset of the files on the root file system can be copied to the second disk. The files that should be copied to the second disk are:

/.cshrc	/.login	/.logout	/.profile	/vmunix
/vmunix1				
/bin/cat	/bin/chgrp	/bin/chmod	/bin/chown	/bin/cp
/bin/cpio	/bin/csh	/bin/dd	/bin/echo	/bin/ln
/bin/ls	/bin/mkdir	/bin/more	/bin/mv	/bin/ps
/bin/pwd	/bin/rm	/bin/rmdir	/bin/sh	/bin/sleep
/bin/sttv	/bin/su	/bin/sync	/bin/tar	/bin/tset
/bin/vi				
/dev/console	/dev/drum	/dev/floppy	/dev/ip0a	/dev/ip0b
/dev/ip0c	/dev/ip0d	/dev/ip0e	/dev/ip0f	/dev/ip0g
/dev/ip0b	/dev/ip1a	/dev/ip1b	/dev/ip1c	/dev/ip1d
/dev/ip1e	/dev/ip1f	/dev/ip1g	/dev/ip1h	/dev/kmem
/dev/md0a	/dev/md0b	/dev/md0d	/dev/md0e	/dev/md0f
/dev/md0g	/dev/md0h	/dev/md1a	/dev/md1b	/dev/md1c
/dev/md1d	/dev/md1e	/dev/md1f	/dev/md1g	/dev/md1h
/dev/mem	/dev/mt1	/dev/nrmt1	/dev/nrtape	/dev/null
/dev/rip0a	/dev/rip0b	/dev/rip0c	/dev/rip0d	/dev/rip0e
/dev/rip0f	/dev/rip0g	/dev/rip0h	/dev/rip1a	/dev/rip1b
/dev/rip1c	/dev/rip1d	/dev/rip1e	/dev/rip1f	/dev/rip1g
/dev/rip1h	/dev/rmd0a	/dev/rmd0b	/dev/rmd0d	/dev/rmd0e
/dev/rmd0f	/dev/rmd0g	/dev/rmd0h	/dev/rmd1a	/dev/rmd1b
/dev/rmd1c	/dev/rmd1d	/dev/rmd1e	/dev/rmd1f	/dev/rmd1g
/dev/rmd1h	/dev/rmt1	/dev/rqic	/dev/swap	/dev/syscon
/dev/systty	/dev/tty	/dev/ttyd1	/dev/ttyd2	/dev/ttyd3
/dev/ttyn0	/dev/ttyn1	/dev/ttyn10	/dev/ttyn11	/dev/ttyn12
/dev/ttyn13	/dev/ttyn14	/dev/ttyn15	/dev/ttyn16	/dev/ttyn17
/dev/ttyn18	/dev/ttyn19	/dev/ttyn2	/dev/ttyn20	/dev/ttyn21
/dev/ttyn22	/dev/ttyn23	/dev/ttyn24	/dev/ttyn25	/dev/ttyn26
/dev/ttyn27	/dev/ttyn28	/dev/ttyn29	/dev/ttyn3	/dev/ttyn30
/dev/ttyn31				
/etc/checklist	/etc/clri	/etc/cshrc	/etc/fsck	/etc/fsdb
/etc/group	/etc/init	/etc/inittab	/etc/link	/etc/mkfs
/etc/mknod	/etc/mnttab	/etc/mount	/etc/passwd	/etc/profile
/etc/reboot	/etc/sys_id	/etc/tar	/etc/telinit	/etc/termcap
/etc/ttytype	/etc/umount	/etc/unlink		

Use **tar** to copy these files. It preserves the modes and ownership of the files. The files in **/dev** should only be "copied" with tar, cpio, or mknod all of which issue a **mknod()** system call rather than actually copy the data. If the first file system on the second disk is mounted on **/mnt** using the command:

```
/etc/mount /dev/md1a /mnt
```

the following commands may be issued to copy these files:

```
cd /

tar -cBf - .cshrc .login .logout .profile vmunix vmunix1 dev | \
(cd /mnt;tar -xBvf -)

cd /bin

tar -cBf - \
cat cbgrp chmod chown cp cpio csh dd echo ln ls \
mkdir more mv ps pwd rm rmdir sh sleep stty su sync tar tset vi \
| (cd /mnt/bin;tar -xBvf -)

cd /etc

tar -cBf - \
checklist clri cshrc fsck fsdb group init inittab link \
mkfs mknod mnttab mount passwd profile reboot sys_id \
tar telinit termcap ttytype umount unlink \
| (cd /mnt/etc;tar -xBvf -)

cd /
```

4.7.3 Creating a Bootable and Using It for Recovery

This method of booting your system should be used only if you cannot boot your system using the above procedures. It loads software from a tape onto the disk, writing over any data that was there before the load. This method is available to systems with release 2.1 or higher. To use this method, you must make a bootable tape to use when your system will not boot.

You can make a bootable tape any time your system is running UNIX. To make a bootable tape, follow these steps:

1. Change to the root of the file system:

   ```
   cd /
   ```

2. Use su to become superuser.

3. Put a tape in the tape drive.

4. Run the **mkboot** program:

   ```
   /etc/mkboot
   ```

You should back up other file systems, such as **usr** on a separate tape using **tar** or **cpio**. **Mkboot** always copies the root. The bootable tape has the following structure:

stand-alone code
root file system

You can use this tape to boot your system and load software onto the disk. Use **Fex** or **Ipfex** to read the tape. **Fex** is used for non-smd type drives and **Ipfex** is used for smd-type drives (such as the Fujitsu Eagle).

The following procedure assumes you cannot boot the machine from the disk and you are willing to write over the disk, destroying the root file system. To read in the tape, follow these steps:

1. Reset the machine by pressing the "reset" button on the monitor. You can also reset by pushing the boot button on the chassis. Some IRIS workstations have a separate box for the mouse communications; it contains a system reset button as well.

2. Put the bootable backup tape in the tape drive.

3. Read the **Fex** program off the tape:

   ```
   tb fex (or ipfex)
   ```

   or, for users with level 3 or higher PROMS:

   ```
   b mt:fex (or ipfex)
   ```

4. Tell **Fex** to create the root file system:

   ```
   t
   ```

5. Specify the defaults that **Fex** offers by typing a carriage-return to each of the following prompts:

   ```
   Tape file (2)?
   Unit (0)?
   File System (a)?
   [Fex types message about block sizes
   it will use for the copy.]
   Type 'go<return> to begin...
     type "'go" and a carriage-return here
   ```

   The following prompts will appear on the display:

   ```
   Copy started...
   ```

```
10 20 30 ....
Tape to Disk Copy complete
Fex 4.4>
```

6. Type the following to re-enter the PROM monitor:

```
q (to quit)
```

Boot the disk in the normal manner.

The /usr file system can be restored while the system is running. To restore the user file system, mount **/usr**, then read in the **/usr** file system by using cpio or tar to read in a tape that contains the **/usr** file system.

4.8 Security

As configured by SGI, the IRIS workstation is, in general, very secure. It is difficult for an unauthorized person to access or change data. One gap in security is that anyone can press the reset button to reboot the system. When the system comes up in single-user mode the "root" shell allows the user to access or change any data on the system. (Of course pressing the reset button while UNIX is running can corrupt the file system.)

Two changes in the configuration files eliminate this security gap. Note that this change reduces the effectiveness of the system's ability to recover from errors in the configuration files, as discussed in Section 4.7.1. First, edit the first line of **/etc/inittab** which, as shipped, reads:

```
is:s:initdefault:
```

to read:

```
is:2:initdefault:
```

Second, edit **/etc/rc** and after the lines:

```
if [ $7 = 2 ]
then
```

add:

```
echo "Checking the File Systems For Consistency"
fsck -q < /dev/console 2>&1 > /dev/console
```

Make both these changes together. They change the configuration of the

system so that when the system is rebooted, instead of going into single-user mode with a root shell (state **s**), it will check the file systems and go directly into multi-user mode (state **2**).

After these changes, no one can use the system without first entering a valid account name and password. If one types Control-C during the **fsck** it will be terminated and the system will enter multi-user mode. This will not corrupt the file systems; however, they will remain unchecked.

Appendix A

System Administration

A. System Administration

The information in this appendix applies to IRIS workstations with PROMs of level 3 or higher; if you have level 3 or higher level PROMs, it supersedes information in Section 6.1 of the IRIS Workstation Guide, version 1.0. All IRIS 1500 and some 1400 workstations have level 3 or higher PROMs. You can find out the level of your boot PROMs by reading the information typed on the console after you push the reset button.

You are responsible for configuring the IRIS workstation to meet local requirements. The sections that follow explain how to boot the IRIS workstation, check the file system, configure UNIX, add new accounts, add ASCII terminals and modems, make backups, shutdown the IRIS workstation and recover from a crash.

This document uses the standard UNIX convention for referring to entries in the UNIX reference manual. The entry name is followed with a section number in parentheses. For example, *cc*(1) refers to the cc manual entry in Section 1 in the *UNIX Programmer's Manual*.

A.1 Startup

A.1.1 Booting the IRIS Workstation

There are two ways to boot an IRIS workstation:

1. Autoboot
2. Manual boot (using the PROM monitor).

When you autoboot, the IRIS sets the hardware to use the primary display. Manual booting allows you to select either type of display monitor (see below).

At power-up, the IRIS workstation is set to enter the IRIS PROM monitor, displaying the prompt:

```
iris>
```

or to automatically boot. The PROM monitor is entered if the `Boot Environment` configuration switches (switches 5-8) on the back panel of the IRIS are set to **0111** where "1" means `Closed` and "0" means `Open`.

If the autoboot switch (switch 4) is `Open` and the boot environment switches are set to select a bootable device (see Table A-1), the IRIS will attempt to boot itself from the file *defaultboot* on the device specified. If the *verbose* configuration switch is `Open` the name of the file the IRIS is attempting to boot will be displayed.

NOTE: If the IRIS workstation is set to boot from a non-existent device, either with the autoboot feature or manually, the system may need to be reset to recover.

Configuration Switches			
Switch	*Name*	*Position*	*Meaning*
1 - 2	Serial line	11[1]	300 baud
		10	19,200 baud
		01	1200 baud
		00	9600 baud
3	Verbose	1	No status reports during power-up testing
		0	Status reports during power-up testing
4	Suppress autoboot	1	Autoboot using the default boot environment (defined below)
		0	Manual Boot
5 - 8	Boot environment	1111	Floppy disk boot
		1110	Disk boot (IRIS 1400)
		1011	Network boot
		0111	PROM Monitor
		0101	Tape boot
		1001	IEEE488 boot
		0110	Eagle Disk boot (IRIS 1500)
		all others	Undefined

Table A-1: IRIS Workstation Configuration Switches

The system can be booted from the PROM monitor by typing **b** (boot), followed by an optional device specifier and an optional file name. If no device specifier is given, the boot environment switches on the back panel are examined. If the switches specify a bootable device, this device becomes the boot device. Otherwise, the boot device defaults to rmd0N: (see below), where N is the file system specified in the disk label on md0. If no filename is given, the file *defaultboot* is assumed. A colon (:) is always the last character in a

1. 1 means `Closed` and 0 means `Open`.

device specifier.

The bootable devices are listed below. The XY portion of the device refers to a device number (X=0,1,2...) and a file system letter (Y=a,b,c...). If the device number is missing and needed, the default is zero. If the file system letter is missing and needed, the label on the specified device is read and the root file system used.

Device Table	
Name	*Device*
mdXY:	Vertex 72 MB disk (1400 only)
rmt:	Streaming tape
mfXY:	Floppy disk
ipXY:	Eagle 440 MB disk (1500 only)
g	IEEE488 (General Purpose Interface Bus)

The syntax for booting from the IEEE488 device is:

```
iris> b g[.x]:[file]
```

where:
`i` is a IEEE488 address (defaults to seven (7) if not present)
`file` is the name of the file to boot

In the simplest case, the command:

```
iris> b
```

would boot the file *defaultboot* from the default device (set in the configuration switches or md0, if none). The command:

```
iris> b mf0a:/stand/picture
```

would look for the file */stand/picture* on file system *a* (usually the root file system is *a*) of floppy drive zero. The command:

```
iris> b md1:vmunix
```

would look for the file *vmunix* on the root file system of Vertex-type drive one.

The IRIS workstation can also be booted over the Ethernet using the SGI XNS protocol. The boot command here is **n** (netboot), followed by a file specifier which consists of *hostname:filename*. The hostname is optional. If it is omitted, the first host to recognize *filename* will respond. For example:

```
iris> n cruncher:/usr/local/boot/goboot
```

will ask the host *cruncher* to send the file */usr/local/boot/goboot* to be booted by the IRIS.

> NOTE: If the IRIS workstation is to be booted from a tape drive, the tape must be in *cpio*(1) format.

A.1.2 Listing the Files on a Device

Since the IRIS workstation can be booted from different environments (hard disks, tape drives, etc.) it can be useful to find the names of the files on a tape or disk before booting. This information can be found with the PROM Monitor. For example:

```
iris> ls /
bin             etc             stand           unix1
defaultboot     lib             tmp             usr
dev             lost+found      unix            version
iris>
```

searches the root file system on the default device and lists its contents. After locating a file, it can be booted explicitly with the b command. For example:

```
iris> b unix
...
```

See Table A-2 for a list of the commands available through the PROM Monitor.

A.2 Boot Verbose Information

If the `Verbose` configuration switch (switch 3) is set to the `Open` position, the IRIS workstation will display the following additional information during system startup:

1. Announce that it is scanning processor memory and display an “X” for each half megabyte of memory and a "." for each non-existent half megabyte of memory
2. Announce that it is clearing processor memory
3. Map processor memory
4. the configuration switch values

This information is intended for diagnostic purposes only. Normally the `Verbose` configuration switch should be set to the `Closed` position.

Command	Description
`h`	Display a list of PROM monitor commands.
`t`	Enter serial interface to host.
`n` *[file]*	Boot *file* over a network. *File* may be in the form of *hostname:filename*.
`b`	Boot *defaultboot* from the default boot device. If the device is the tape drive, the file must be in *cpio* format.
`b` *[file]*	Boot *file* on the default boot device. The default boot device is determined from the configuration switches (see Table A-1).
`b` *[dev]:[file]*	Boot *file* on device *dev*.
`ls` *[dev]:[pathname]/**	List the contents of directory *pathname* on device *dev*. If **ls** is used with no arguments, the monitor provides a list of the available devices.
`b` *[dev]*:	Boot file *defaultboot* from specified device.
`r`	Restart the PROM monitor.

Table A-2: PROM Monitor Commands

Appendix B

Stand-alone Disk Formatting

Fex

Disk Formatter And Exerciser

V4.4

Introduction

Fex is a Formatter and EXerciser (thus its name) for winchester disk drives on the IRIS 1400. It will handle one or two drives, and supports any combination of VERTEX V170 and ATASI 3046 drives. Fex also manages disk labels, which are software labels describing the type of drive, how the drive is partitioned logically, and bad track information. Fex will also copy all or portions of disks from one drive to another and it will copy tapes to disk. Fex can also be used to format and exercise floppies (see **set**unit below). *On a GL2 machine only*:

Fex and **ipfex** do not work yet on GL2 machines. Set up a dumb terminal as the console (refer to Chapter 6 of the IRIS Workstation Guide for a discussion of proper configuration switches.)

Theory of operation

Most input to fex is performed by single characters, with NO <return>s, except for numbers and strings which must end with a <return>. When single characters are expected, fex will complete the command or request immediately, usually asking for further input. Strings and numbers may be edited using <BACK-SPACE> to delete the previous character, and <CTRL-U> to delete all characters back to the last prompt. Use a <DEL> at any time to return to the Fex 4.4> top level command prompt. When fex starts up its herald will look like:

SGI Formatter/Exerciser 4.4 September 24, 1984
Initialize drive 0
Name: <drive name>, Serial: <drive serial #>
Fex 4.4 >

If the drive has no label, (i.e., hasn't been run through fex or the manufacturing disk exerciser), the start up will be:

No label--Type of drive (vertex)?

If you answer "**?**<return>" you will be given a list of the known drives (vertex or atasi for now). Enter a "<return>" to accept the default (vertex), or

enter "**atasi**<return>" if Drive 0 is an Atasi. You will then get:

```
    Using default label
Fex 4.4 >
```

Which reminds the user that the drive name (e.g., Beta Release 1.7, etc.), serial number, and bad block information needs to be entered.

At the **Fex 4.4>** prompt, use "h" to obtain a help message, which will look like:

```
Fex 4.4 > Help--Commands are:
  badblock - enter bad block edit mode
  copy     - copy data
  exercise - run drive read/write/seek tests
  format   - format the selected drive
  help     - print this message
  initialize - initialize drive & read label
  mapbad   - map out a bad track
  quit     - quit; return to IRIS monitor
  set      - set miscellaneous variables
  tape     - tape copy to disk utility
```

To select a command, enter the first character of the command. The next section describes each command in detail.

Fex will be used in the field for several purposes. The customer will use fex to restore a disk from tape and map out bad tracks which develop over time under normal operating conditions. The command **mapbad** is provided for this purpose, and should be needed very rarely. Another use of fex will be to reformat a disk if its formatting information is lost. This can happen if a power supply should fail while the disk is running, or a power failure should occur during a disk write operation.

When a disk is formatted, all data on the disk is destroyed, so this should only be done with a new disk, or with a complete disk backup, either from tape or from another disk.

To operate on Drive #1 rather than Drive #0 (the default), use the **set**unit command (see below).

Commands/Tests

badblock

This command will prompt:

```
Bad Block edit, type h for help
 bb>
```

and "h" will produce the message:

bb > Help—choose one of
add bad blocks
clear bad block list
edit list
print list
quit
setup alternates
zap alternate assignments

Use **add** to enter new bad blocks to the bad block list tor the currently selected drive (see **set** below). The entry format is cylinder/head, where the cylinder and head are read directly from the track list provided by the drive manufacturer. A session might look like:

bb> Add new entries. Mode cyl/hd(/sec), end with a blank line:
bb add: 45/5
bb add: 103/0
bb add: 200/0
bb add: 450/3
bb add:
bb>

End the bad block list with a blank line (i.e., simply press "<return>"). If the "/head" is left off, the previous bad block's head will be used, and printed out as if you had typed it. For ease of entry, a "." may be used in place of the "/" between the cylinder and the head, facilitating the use of the numeric key pad at the right of the keyboard. The <ENTER> key on the number pad is the same as <return>.

Use **clear** to delete the current list. This is useful if you've entered the wrong list, or the list is wrong because it was copied from another drive.

Use **edit** to repair any typographical errors. A session (using the above entered bad blocks) would look something like:

bb> Edit bad blocks:
For each bad block, press 'space' to keep, 'd' to delete, 'q' to quit...
bb edit 45/5? Kept
bb edit 103/0? Deleted
bb edit 200/0? Kept
bb edit 450/3? Kept
bb>

In this example, the second bad block (103/0) was deleted ("d" typed) from the bad block list, and the rest were kept ("<SPACE>" typed).

Use **print** to print the current list of bad blocks. This should be done prior to formatting to verify that the list is correct. This command is the same as **set**badblocks The list shows only the bad tracks unless the **verbose** flag is set (see **set** below), in which case the assigned alternate

tracks are also listed.

Use **quit** to return to the **Fex 4.4>** prompt and leave bad block mode.

Use **setup** to assign alternate tracks to newly entered bad tracks. This is done automatically when the drive is formatted, but can be done by hand to see the assigned alternates.

Use **zap** to clear the assigned alternate associations. This should be done if a bad block was added to an existing bad block list and the drive is going to be reformatted. Normally, associations between bad tracks and alternates are preserved to retain data integrity, but if the drive is going to be reformatted, the alternate association should be redone for layout efficiency.

copy

This command copies data between drives. It can be used to copy data from a good system disk to a newly formatted drive. **Be careful not to write to your good system disk!** A copy session will look something like:

```
Fex 4.4 > Copy Data From:
Disk Unit (0)=
Disk address (cyl/hd(/sec))(0/0/0)?
To:
Disk Unit (1)=
Initialize drive 1
  Name <some name>, Serial: <serial>
Disk address (cyl/hd(/sec))(0/0/0)?
 # of sectors per transfer (chunk)(119)?
 # of chunks to transfer (970)?
Verify? Yes
.....................................................................................................
.....................
Copy complete!
Label on drive 1 needs updating... do it? Yes
Fex 4.4 >
```

The numbers in parentheses are the defaults which will be used if a <return> is typed. The From and To units may be the same, and specify which drive to read and write respectively. (If either of the drives has not been initialized, it will be initialized when the unit is specified, thus the “Initialize drive 1” message in the above example.) The From and To disk addresses are the starting point for the transfer. The # of sectors per transfer will default to one cylinder, and the # of chunks to transfer will default to the whole disk. If you answer “y” to Verify, each chunk written will be read and verified against the original data. Each “.” represents 10 chunks, and thus there are 97 of them in this example. When the copy completes, the target drive label will need to be updated if it was over-written by the copy (which it was in

this case); answer “y”.

exercise

This command enters the disk exerciser portion of **Fex 4.4>**. Its prompt and help message are:

```
Fex 4.4 > Exercise
  Drive: vertex Unit=0, (970+17/7/17(512)) ILV=1
***Warning: set write lock off to scribble on the disk
Which exercise? Help--Choose from:
  complete write/read multi pass/multi pattern
  error display/reset
  quit
  random reads
Which exercise?
```

You will get the warning about the writelock only if the software write lock is on. Use **set**writelock to turn it off if you really want to scribble on a disk.

Most of these exercises are used for diagnostic purposes to qualify and time a new disk drive. Only the **complete** and **error display** commands should be used in the field. The **complete** test will prompt:

```
Which exercise? Complete Exercise -- track writes and reads
Repeat how many times? Forever
Alternate units? No
Unit 0: Pattern 0xB1B6DB6D
 10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160
170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320
330 340 350 360 370 380 390 400 410 420 430 440 450 460 470 480
490 500 510 520 530 540 550 560 570 580 590 600 610 620 630 640
650 660 670 680 690 700 710 720 730 740 750 760 770 780 790 800
810 820 830 840 850 860 870 880 890 900 910 920 930 940 950 960
970 980
Pass 1 Unit 0 bad 0 retries 0 total bad 0
Unit 0: Pattern 0xB1F6DB6D
```

To the “Repeat” question, press “<return>” to select “Forever”, otherwise enter the number of loops desired and press “<return>”. To the “Alternate units” question, if you answer “n”, only the currently selected drive will be exercised, otherwise, both drives will be exercised. Various data patterns will be written, read and verified sequentially to the whole disk (or disks). If any errors are encountered, messages will be printed, and a summary of good and bad passes will be output at the end of each pass of the test.

After exercising a disk, use **error**display/reset to print out the cumulative error information from the exercise. If bad spots are reported which are not in the bad block lists for each drive, add them to the bad block lists, and reformat each drive as necessary. Before retesting the

drive using the **complete** test remember to reset the error information for a correct reading of the disk errors.

format

This command will format the currently selected disk drive. When selected, it will prompt:

```
Fex 4.4> Format disk.
***WARNING -- ALL DATA ON UNIT 0 WILL BE LOST!!!
  Drive: vertex Unit=0 (970+17/7/17(512)) ILV=1
***WARNING -- No bad blocks!!!
Type 'go<return>' to start...
```

The Unit will be the currently selected unit (see set below). The drive information should correspond properly to the type of drive in the system (otherwise, the label was set up improperly, and should be changed using **set label**). The "No bad blocks" warning will only be printed if no bad blocks have been entered for the drive (see **badblock** above). When you type "go<return>", the output will be

```
Starting format...
 10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160
170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320
330 340 350 360 370 380 390 400 410 420 430 440 450 460 470 480
490 500 510 520 530 540 550 560 570 580 590 600 610 620 630 640
650 660 670 680 690 700 710 720 730 740 750 760 770 780 790 800
810 820 830 840 850 860 870 880 890 900 910 920 930 940 950 960
970 980
Formatting bad tracks...
Writing label...
Formatting complete.
```

Formatting will rarely produce any errors, so it is wise to run "exercise complete" after every format.

help

This command produces the help message displayed above.

initialize

This command reads the label from the currently selected drive, overwriting the in-core copy. It will prompt:

```
Fex 4.4> Initialize drive 0
***Warning: in-core label newer than disk copy.
***Clobber in-core label? y
  Name: drive name, Serial: #####
```

If you have modified the in-core label, either by set label or changing the bad block list, you will receive the warning shown above. Answer "y< return >" if you want to undo your label modifications and return to the label on the disk. If you answer anything but "y", the in-core label will be preserved, and the initialize aborted. If the drive has no label at all, you will get:

Fex 4.4> Initialize drive 0
No label--Type of drive (vertex)?
Using default label

At the type of drive prompt, type a "?" to obtain a list of the known drive types, or a <return > to use the default type (as shown in the parentheses). If the label on the drive is unreadable (i.e., it has not even been formatted), you may get a disk error message printed before the "No label" line. This is normal, and indicates that the drive needs to be formatted,

mapbad

This command will add a bad track to an active disk. That is, if a bad spot develops on the disk while it is in use, this command will substitute a good track for the bad one, and copy data from the bad track to the new alternate. It will prompt:

Fex 4.4> Map a bad track
Name: drive name. Serial: ###
Bad track: Cylinder: 123 Head: 5
Reading old data into memory...
Re-formatting bad track and its alternate...
Re-writing data to new track...
Re-writing label
Track remap complete.

Warning: Insure that the drive label has all of the known bad blocks (from the list on the drive) recorded. If NOT - set the blocks first and then issue the mapbad command.

If there is a problem reading a sector of the specified bad track, you will get error messages after the "Reading old data" line. If after 10 failures at reading a sector, you will be prompted with:

Old data read error: Retry/Skip/Quit/Verbose?

Answer "r", "s", "q", or "v" as appropriate. You may want to retry a few times, then skip the bad sector. If you skip a sector, remember that its data will be lost, which may be the only way to **solve** the problem.

Once the old track has been read, the old and new tracks are reformatted, and the saved data is written to the new track. If the data rewrite produces errors, it means that there are problems with the newly assigned alternate track. This probably indicates a more severe drive problem, and the whole drive should be re-formatted and exercised, and perhaps exchanged. NOTE: re-formatting will erase all data, so perform a full backup first.

quit

This command exits from fex. It prompts

Fex 4.4> Quit Label on drive 0 needs updating... do it? y
--confirm quit with "y": y

If one or both of the drives need their labels updated, you will be asked if they should be re-written; answer yes. Finally, confirm the quit with "y<return>" and you will be returned to the IRIS> prom monitor prompt.

set

This command is a general purpose variable and parameter modifier. Its prompt and help message are:

Fex 4.4 > Set ? Set commands are:

badblocks - display current drive bad block list
defaults - display settings
label - set up the label
unit - select unit for testing (0/1/f)
verbose - verbose (on/off)
writelock - write lock switch (on/off)

Set **badblocks** displays the list of bad blocks for the currently selected drive. This is equivalent to the "print" option of the badblock command.

Set **defaults** displays the current list of global program defaults, and the label for the currently selected drive.

Set **label** allows the user to enter the information for the disk label, eg. the Serial number of the disk and the type of the disk.

Set **unit** selects the current disk drive unit ("0" or "1"), or the floppy ("f") drive. Only one floppy drive is supported.

Set **verbose** turns on or off verbose (debugging) output.

Set **writelock** turns on or off software write lock of the disks. Writelock must be turned off before exercise will write any disk blocks.

tape

This command is the special purpose command to build or rebuild the disk from a master configuration tape. It will prompt:

Fex 4.4> Tape to Disk copy
Tape file (2)?
Unit (0)?
File System (a)?
Copying 8925K in 119 blk chunks from tape file 2 to md0a
Type 'go<return>' to begin...

When you type "go<return>", the output will be:

```
Copy started...
10  20  30  40  50  60  70  80  90 100 110 120 130 140 150
Tape to Disk Copy complete
Fex 4.4>
```

In this example, all of the default parameters were used. The root file system on the master configuration tape is in file 2. The loading of the root file system is complete and afterwards, unix can be booted from the disk.

The disk label must be set up correctly before the tape copy routine is used.

Error Messages

The above discussion listed most of the warning messages that fex produces. Other warnings are self explanatory.

There are many error messages output by fex and we've tried to make them as self-explanatory as possible.

Ipfex

Interphase Disk Formatter And Exerciser

V2.0

Introduction

Ipfex is a Formatter and Exerciser (thus its name) for SMD disk drives on the IRIS 1500. It will handle one or two drives, and supports any combination of FUJITSU 2351A (eagles) and Fujitsu 2312 drives. Ipfex also manages disk labels, which are software labels describing the type of drive, how the drive is partitioned logically, and bad track information. Ipfex will also copy all or portions of disks from one drive to another and it will copy tapes to disk. Ipfex can also be used to format and exercise floppies (see *set unit* below).

Theory of operation

Most commands and data to ipfex should not be followed with carriage returns except for numbers and strings which must end with a carriage return. When single character commands are expected, ipfex will complete the command or request immediately, usually asking for further input. Strings and numbers may be edited using backspace key to delete the previous character, and Control-U to delete all characters back to the last prompt. A DELete key pressed at any time will return to the top level command prompt:

Ipfex 2.0 >

When ipfex starts up its herald will look like:

SGI Formatter/Exerciser 2.0 September 24, 1984
Initialize drive 0
Name: <drive name>, Serial: <drive serial #>
Ipfex 2.0>

If the drive has no label, (i.e., hasn't been run through ipfex or the manufacturing disk exerciser), the start up will be:

No label—Type of drive (eagle)?

If you answer "**?**<return>" you will be given a list of the known drives (eagle or Fujitsu 2312 for now). Enter a "<return>" to accept the default (eagle), or enter "**2312**<return>" if Drive 0 is an Fujitsu 2312. You will then get:

Using default label
Ipfex 2.0>

Which reminds the user that the drive name (e.g., Beta Release 1.7, etc.), serial number, and bad block information needs to be entered.

At the **Ipfex 2.0>** prompt, use "h" to obtain a help message, which will look like:

```
Ipfex 2.0> Help--Commands are:
  badblock - enter bad block edit mode
  copy     - copy data
  exercise - run drive read/write/seek tests
  format   - format the selected drive
  help     - print this message
  initialize - initialize drive & read label
  mapbad   - map out a bad track
  quit     - quit; return to IRIS monitor
  set      - set miscellaneous variables
  tape     - tape copy to disk utility
```

To select a command, enter the first character of the command. The next section describes each command in detail.

Ipfex will be used in the field for several purposes. The customer will use ipfex to restore a disk from tape and map out bad tracks which develop over time under normal operating conditions. The command *mapbad* is provided for this purpose, and should be needed very rarely. Another use of ipfex will be to reformat a disk if its formatting information is lost. This can happen if a power supply should fail while the disk is running, or a power failure should occur during a disk write operation.

When a disk is formatted, all data on the disk is destroyed, so this should only be done with a new disk, or with a complete disk backup, either from tape or from another disk.

To operate on Drive #1 rather than Drive #0 (the default), use the *set unit* command (see below).

Commands/Tests

badblock

This command will prompt:

```
Bad Block edit, type h for help
 bb>
```

and "h" will produce the message:

```
bb> Help--choose one of
 add bad blocks
 clear bad block list
 edit list
 print list
 quit
 setup alternates
 zap alternate assignments
```

Use *add* to enter new bad blocks to the bad block list tor the currently selected drive (see *set* below). The entry format is cylinder/head, where the cylinder and head are read directly from the track list provided by the drive manufacturer. A session might look like:

```
bb> Add new entries. Mode cyl/hd(/sec), end with a blank line:
bb add: 45/5
bb add: 103/0
bb add: 200/0
bb add: 450/3
bb add:
bb>
```

End the bad block list with a blank line (i.e., simply press "<return>"). If the "/head" is left off, the previous bad block's head will be used, and printed out as if you had typed it. For ease of entry, a "." may be used in place of the "/" between the cylinder and the head, facilitating the use of the numeric key pad at the right of the keyboard. The <ENTER> key on the number pad is the same as <return>.

Use *clear* to delete the current list. This is useful if you've entered the wrong list, or the list is wrong because it was copied from another drive.

Use *edit* to repair any typographical errors. A session (using the above entered bad blocks) would look something like:

```
 bb> Edit bad blocks:
For each bad block, press 'space' to keep, 'd' to delete, 'q' to quit...
  bb edit 45/5? Kept
  bb edit 103/0? Deleted
  bb edit 200/0? Kept
  bb edit 450/3? Kept
 bb>
```

In this example, the second bad block (103/0) was deleted ("d" typed) from the bad block list, and the rest were kept ("<SPACE>" typed).

Use *print* to print the current list of bad blocks. This should be done prior to formatting to verify that the list is correct. This command is the same as *set badblocks* . The list shows only the bad tracks unless the *verbose* flag is set (see *set* below), in which case the assigned alternate

tracks are also listed.

Use *quit* to return to the **Ipfex 2.0>** prompt and leave bad block mode.

Use *setup* to assign alternate tracks to newly entered bad tracks. This is done automatically when the drive is formatted, but can be done by hand to see the assigned alternates.

Use *zap* to clear the assigned alternate associations. This should be done if a bad block was added to an existing bad block list and the drive is going to be reformatted. Normally, associations between bad tracks and alternates are preserved to retain data integrity, but if the drive is going to be reformatted, the alternate association should be redone for layout efficiency.

copy

This command copies data between drives. It can be used to copy data from a good system disk to a newly formatted drive. **Be careful not to write To your good system disk!** A copy session will look something like:

```
Ipfex 2.0> Copy Data
From:
Disk Unit (0)=
Disk address (cyl/hd(/sec))(0/0/0)?
To:
Disk Unit (1)=
Initialize drive 1
  Name <some name>, Serial: <serial>
Disk address (cyl/hd(/sec))(0/0/0)?
# of sectors per transfer (chunk)(880)?
# of chunks to transfer (842)?
Verify? Yes
.....
.....
Copy complete!
Label on drive 1 needs updating... do it? Yes
Ipfex 2.0>
```

The numbers in parentheses are the defaults which will be used if a <return> is typed. The From and To units may be the same, and specify which drive to read and write respectively. (If either of the drives has not been initialized, it will be initialized when the unit is specified, thus the "Initialize drive 1" message in the above example). The From and To disk addresses are the starting point for the transfer. The # of sectors per transfer will default to one cylinder, and the # of chunks to transfer will default to the whole disk. If you answer "v" to Verify, each chunk written will be read and verified against the original data. Each "." represents 10 chunks, and thus there are 97 of them in this example. When the copy completes, the target drive label will

need to be updated if it was over-written by the copy (which it was in this case); answer "y".

exercise

This command enters the disk exerciser portion of **Ipfex 2.0>**. Its prompt and help message are:

```
Ipfex 2.0> Exercise
  Drive: eagle Unit=0, (837+5/20/44(512)) ILV=1
***Warning: set write lock off to scribble on the disk
Which exercise? Help--Choose from:
  complete write/read multi pass/multi pattern
  error display/reset
  quit
  random reads
Which exercise?
```

You will get the warning about the writelock only if the software write lock is on. Use *set writelock* to turn it off if you really want to scribble on a disk.

Most of these exercises are used for diagnostic purposes to qualify and time a new disk drive. Only the *complete* and *error display* commands should be used in the field. The *complete* test will prompt:

```
Which exercise? Complete Exercise -- track writes and reads
Repeat how many times? Forever
Alternate units? No
Unit 0: Pattern 0xB1B6DB6D
 10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160
170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320
330 340 350 360 370 380 390 400 410 420 430 440 450 460 470 480
490 500 510 520 530 540 550 560 570 580 590 600 610 620 630 640
650 660 670 680 690 700 710 720 730 740 750 760 770 780 790 800
810 820 830
Pass 1 Unit 0 bad 0 retries 0 total bad 0
Unit 0: Pattern 0xB1F6DB6D
```

To the "Repeat" question, press "<return>" to select "Forever", otherwise enter the number of loops desired and press "<return>". To the "Alternate units" question, if you answer "n", only the currently selected drive will be exercised, otherwise, both drives will be exercised. Various data patterns will be written, read and verified sequentially to the whole disk (or disks). If any errors are encountered, messages will be printed, and a summary of good and bad passes will be output at the end of each pass of the test.

After exercising a disk, use *error display/reset* to print out the cumulative error information from the exercise. If bad spots are reported which are not in the bad block lists for each drive, add them to the bad block lists, and reformat each drive as necessary. Before retesting the drive

using the *complete* test remember to reset the error information for a correct reading of the disk errors.

format

This command will format the currently selected disk drive. When selected, it will prompt:

```
Ipfex 2.0> Format disk.
***WARNING - ALL DATA ON UNIT 0 WILL BE LOST!!!
  Drive: eagle Unit=0 (837+5/20/44(512)) ILV=1
***WARNING - No bad blocks!!!
Type 'go<return>' to start...
```

The Unit will be the currently selected unit (see *set* below). The drive information should correspond properly to the type of drive in the system (otherwise, the label was set up improperly, and should be changed using *set label*). The "No bad blocks" warning will only be printed if no bad blocks have been entered for the drive (see *badblock* above). When you type "go<return>", the output will be:

```
Starting format...
 10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160
170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320
330 340 350 360 370 380 390 400 410 420 430 440 450 460 470 480
490 500 510 520 530 540 550 560 570 580 590 600 610 620 630 640
650 660 670 680 690 700 710 720 730 740 750 760 770 780 790 800
810 820 830 840
Formatting bad tracks...
Writing label...
Formatting complete.
```

Formatting will rarely produce any errors, so it is wise to run "exercise complete" after every format.

help

This command produces the help message displayed above.

initialize

This command reads the label from the currently selected drive, overwriting the in-core copy. It will prompt:

```
Ipfex 2.0> Initialize drive 0
***Warning: in-core label newer than disk copy.
***Clobber in-core label? y
Name: drive name, Serial: #####
```

If you have modified the in-core label, either by *set label* or changing the bad block list, you will receive the warning shown above. Answer "y<return>" if you want to undo your label modifications and return to the label on the disk. If you answer anything but "y", the in-core label will be preserved, and the initialize aborted. If the drive has no label at all, you will get:

Ipfex 2.0> Initialize drive 0
No label--Type of drive (eagle)?
Using default label

At the type of drive prompt, type a "?" to obtain a list of the known drive types, or a < return > to use the default type (as shown in the parentheses). If the label on the drive is unreadable (i.e., it has not even been formatted), you may get a disk error message printed before the "No label" line. This is normal, and indicates that the drive needs to be formatted.

mapbad

This command will add a bad track to an active disk. That is, if a bad spot develops on the disk while it is in use, this command will substitute a good track for the bad one, and copy data from the bad track to the new alternate. It will prompt:

Ipfex 2.0> Map a bad track
Name: drive name, Serial: ###
Bad track: Cylinder: 123 Head: 5
Reading old data into memory...
Re-formatting bad track and its alternate...
Re-writing data to new track...
Re-writing label
Track remap complete.

WARNING: Insure that the drive label has all of the known bad blocks (from the list on the drive) recorded. If NOT - set the blocks first, then issue the mapbad command. If there is a problem reading a sector of the specified bad track, you will get error messages after the "Reading old data" line. If after 10 failures at reading a sector, you will be prompted with:

Old data read error: Retry/Skip/Quit/Verbose?

Answer "r", "s", "q", or "v" as appropriate. You may want to retry a few times, then skip the bad sector. If you skip a sector, remember that its data will be lost, which may be the only way to *solve* the problem.

Once the old track has been read, the old and new tracks are reformatted, and the saved data is written to the new track. If the data rewrite produces errors, it means that there are problems with the newly assigned alternate track. This probably indicates a more severe drive problem, and the whole drive should be re-formatted and exercised, and perhaps exchanged. NOTE: re-formatting will erase all data, so perform a full backup first.

quit

This command exits from ipfex. It prompts:

Ipfex 2.0> Quit Label on drive 0 needs updating... do it? y
--confirm quit with "y": y

If one or both of the drives need their labels updated, you will be asked if they should be re-written; answer yes. Finally, confirm the quit with "y<return>" and you will be returned to the IRIS> prom monitor prompt.

set

This command is a general purpose variable and parameter modifier. Its prompt and help message are:

Ipfex 2.0 > Set ? Set commands are:
badblocks - display current drive bad block list
defaults - display settings
label - set up the label
unit- select unit for testing (0/1/f)
verbose - verbose (on/off)
writelock - write lock switch (on/off)

Set *badblocks* displays the list of bad blocks for the currently selected drive. This is equivalent to the "print" option of the *badblock* command.

Set *defaults* displays the current list of global program defaults, and the label for the currently selected drive.

Set *label* allows the user to enter the information for the disk label, eg. the Serial number of the disk and the type of the disk.

Set *unit* selects the current disk drive unit ("0" or "1"), or the floppy ("f") drive. Only one floppy drive is supported.

Set *verbose* turns on or off verbose (debugging) output.

Set *writelock* turns on or off software write lock of the disks. Writelock must be turned off before exercise will write any disk blocks.

tape

This command is the special purpose command to build or rebuild the disk from a master config tape. It will prompt:

Ipfex 2.0> Tape to Disk copy
Tape file (2)?
Unit (0)?
File System (a)?
Copying 14080K in 880 blk chunks from tape file 2 to ip0a
Type 'go<return>' to begin...

When you type "go<return>", the output will be:

```
Copy started...
2 4 6 8 10 12 14 16
Tape to Disk Copy complete
Ipfex 2.0>
```

In this example, all of the default parameters were used. The root file system on the master configuration tape is in file 2. The loading of the root file system is complete and afterwards, unix can be booted from the disk.

The disk label must be set up correctly before the tape copy routine is used.

Error Messages

The above discussion listed most of the warning messages that ipfex produces. Other warnings are self explanatory.

There are many error messages output by ipfex and we've tried to make them as self-explanatory as possible.

Appendix C

New Manual Pages

NAME

adb – debugger

SYNOPSIS

adb [**–w**] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is usually an executable program file, preferably containing a symbol table; if not, then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is **a.out**. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *Corfil* is **core**.

Requests to *adb* are read from the standard input and responses are to the standard output. If the **–w** flag is present, then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

To exit *adb* use $q or $Q or Control-d.

In general requests to *adb* are of the form

[*address*] [, *count*] [*command*] [;]

If *address* is present, then *dot* is set to address. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged, then addresses are interpreted in the usual way in the address space of the subprocess. If the operating system is being debugged either post-mortem or using the special file **/dev/kmem** to interactive examine and/or modify memory, the maps are set to map the kernel virtual addresses. For further details of address mapping see Addresses.

Expressions

. The value of *dot*.

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer A number. The prefix 0 (zero) forces interpretation in octal radix; the prefixes 0d and 0D force interpretation in decimal radix; the prefixes 0x and 0X force interpretation in hexadecimal radix. Thus 020 = 0d16 = 0x10 = sixteen. If no prefix appears, then the

default radix is used; see the $d command. The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF with the obvious values. Note that a hexadecimal number whose most significant digit would otherwise be an alphabetic character must have a 0x (or 0X) prefix (or a leading zero if the default radix is hexadecimal).

'cccc' The ASCII value of up to 4 characters. \ may be used to escape an '.

<name The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see Variables) named by single letters or digits. If name is a register name, then the value of the register is obtained from the system header in *corfil*. The register names are those printed by the $r command.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. \ may be used to escape other characters. The value of the symbol is taken from the *symbol* table in *objfil*. An initial _ or ~ will be prepended to *symbol* if needed.

_symbol In C, the "true name" of an external symbol begins with _ . It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

(exp) The value of the expression *exp*.

Monadic operators:

**exp* The contents of the location addressed by *exp* in *corfil*.

@exp The contents of the location addressed by *exp* in *objfil*.

–exp Integer negation.

~exp Bitwise complement.

#exp Logical negation.

Dyadic operators are left associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1–e2 Integer subtraction.

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1#e2 *E1* rounded up to the next multiple of *e2*.

Commands

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands "?" and "/" may be followed by "*"; see Addresses for further details.)

?f Locations starting at *address* in *objfil* are printed according to the format *f*. *Dot* is incremented by the sum of the increments for each format letter (q.v .).

/f Locations starting at *address* in *corfil* are printed according to the format *f,* and *dot* is incremented as for "?".

=f The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format "?" is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format, *dot* is incremented by the amount given for each format letter. If no format is given, then the last format is used. The format letters available are as follows:

i	*n*	Disassemble the addressed instruction.
o	2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O	4	Print 4 bytes in octal.
q	2	Print in signed octal.
Q	4	Print long signed octal.
d	2	Print in decimal.
D	4	Print long decimal.
x	2	Print 2 bytes in hexadecimal.
X	4	Print 4 bytes in hexadecimal.
u	2	Print as an unsigned decimal number.
U	4	Print long unsigned decimal.
f	4	Print the 32-bit value as a floating point number.
F	8	Print double (SGI *long float*) floating point.
b	1	Print the addressed byte in octal.
c	1	Print the addressed character.
C	1	Print the addressed character using the standard escape convention where control characters are printed as ^X and the delete character is printed as ^?.
s	*n*	Print the addressed characters until a zero character is reached.
S	*n*	Print a string using the ^X escape convention (see **C** above). The *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see *ctime*(3C)).

a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
/ global data symbol
? global text symbol
= global absolute symbol

p 4 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

r 0 Print a space.

n 0 Print a newline.

"..." 0 Print the enclosed string.

^ *Dot* is decremented by the current increment. Nothing is printed.

+ *Dot* is incremented by 1. Nothing is printed.

– *Dot* is decremented by 1. Nothing is printed.

newline
Repeat the previous command with a *count* of 1.

[?/]l *value mask*
Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used, then the match is for 4 bytes at a time instead of 2. If no match is found, then *dot* is unchanged; otherwise, *dot* is set to the matched location. If *mask* is omitted, then –1 is used.

[?/]w *value ...*
Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m *b1 e1 f1*[**?/**]
New values for (*b1*, *e1*, *f1*) are recorded. If less than three expressions are given, then the remaining map parameters are left unchanged. If the "?" or "/" is followed by "*", then the second segment (b2, e2, f2) of the mapping is changed. If the list is terminated by "?" or "/", then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, "/m?" will cause "/" to refer to *objfil*.)

>*name*
Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following "!".

$*modifier*
Miscellaneous commands. The available *modifiers* are:

<*f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If *f* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable *9* before the first command in *f* is executed.

<<*f* Similar to **<** except it can be used in a file of commands without causing the file to be closed. Variable *9* is saved during the execution of this command, and restored when it completes. There is a (small) finite limit to the number of **<<** files that can be open at once.

>*f* Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.

? Print process ID, the signal which caused stoppage or termination, as well as the registers as **$r**. This is the default if *modifier* is omitted.

r Print the general registers and the instruction addressed by pc. Dot is set to pc.

b Print all breakpoints and their associated counts and commands.

c C stack backtrace. If address is given, then it is taken as the address of the current frame (instead of a7). If count is given, then only the first count frames are printed.

d Set the default radix to decimal.

w Set the page width for output to address (default 80).

s Set the limit for symbol matches to address (default 255).

x Interpret integer input in hexadecimal radix.

q Exit from adb.

v Print all non zero variables in octal, .

m Print the address map.

:*modifier*

Manage a subprocess. Available modifiers are:

b*c* Set breakpoint at *address*. The breakpoint is executed *count*–1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command is omitted or sets *dot* to zero then the breakpoint causes a stop.

d Delete breakpoint at *address*.

r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point, *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

c*s* The subprocess is continued with signal *s* (see *signal*(2)). It *address* is given, then the subprocess is continued at this address. It no signal is specified, then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

s Single step the subprocess *count* times.

k The current subprocess, if any, is terminated.

Variables

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows:

0 The last value printed.
1 The last offset part of an instruction source.
9 The count on the last $< or $<< command.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a **core** file, then the following values are set from *objfil*:

b The base address of the data segment.
d The data segment size.
e The entry point.
m The "magic" number (0407, 0410).
s The stack segment size.
t The text segment size.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1*, *e1*, *f1* and (*b2*, *e2*, *f2* and the file address corresponding to a written *address* is calculated as follows:

$b1 \leq address < e1 \Rightarrow file\ address = address + f1 - b1$, otherwise,

$b2 \leq address < e2 \Rightarrow file\ address = address + f2 - b2$,

otherwise, the requested *address* is not legal. If a **?** or / is followed by an *, then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, then for that file *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

So that *adb* may be used on large files all appropriate values are kept as

signed 32-bit integers.

EXAMPLE

```
adb obj1
```

will invoke *adb* with the executable object "obj1"; when *adb* responds with:

```
ready
```

the request:

```
main, 10?ia
```

will cause 16 (10hex) instructions to be printed in assembly code, starting from location "main".

FILES

a.out
core

SEE ALSO

a.out(4), core(4)

DIAGNOSTICS

Adb when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

Use of # for the unary logical negation operator is peculiar.

When stopping at the entry to a function, the breakpoint should be placed at *routine+4*, rather than at *routine*. This causes the link to be done before the breakpoint, and makes a stacktrace work better.

There is no way to clear all breakpoints.

A floating point number cannot be written into memory (i.e., using the *w* command).

This version of *adb* is capable only of recognizing *hexadecimal* and *decimal* input radices.

NAME

arch – display a simulated architectural model

SYNOPSIS

/usr/people/demos/arch

DESCRIPTION

Arch displays a model of an imaginary city block. Operator controls adjust the viewing position and select one or several structures for display.

Most user interface is through the mouse buttons. Each non-zero combination of mouse buttons causes a different viewpoint movement. The mouse valuators are ignored.

In the default case, a single building is displayed. Type an *s* to turn on (or off) wireframe buildings that complete the block. Type *f* To cover (or uncover) the building surfaces. Type *r* to initialize the program and *h* for a help menu that lists the mouse functions.

AUTHOR.

Gary Tarolli

NAME

car – display of car body in wireframe & surface form

SYNOPSIS

/usr/people/demos/car

DESCRIPTION

A car body is viewed in wireframe form. Once the car is positioned, the surface patches are filled in to give a rendered image of the car.

Mousebuttons	Function
left	mouse x-motion controls view azimuth, mouse y-motion controls view elevation
left & middle	mouse x-motion controls view distance
middle & right	toggle between wireframe & rendered images

The appropriate viewing action taken as a result of mousebutton control is briefly flashed in the in the lower left/right hand comers of the screen.

HARDWARE REQUIREMENTS

Car requires 12 bitplanes.

NAME

cc, pc, f77 – C, Pascal and FORTRAN compilers tor the 68000

SYNOPSIS

cc [options] files ...
pc [options] files ...
f77 [options] files ...

DESCRIPTION

Cc is the UNIX C, Pascal and Fortran compiler for the 68000. It is also available under the names *f77* and *pc*. The names are synonymous *except* during the linking phase, when it is used to create the appropriate run time environment. *Cc* accepts many types of input files, determined by the file's suffix. The highest form of input is language source — C (*.c*), Pascal (*.p*) or FORTRAN (*.f*). These are translated to the language's intermediate format (68000 assembler (*.s*), in the case of C, and a special object format (*.j*), in the case of FORTRAN and Pascal), then to UNIX object files (*.o*), and finally to an executable file, usually called *a.out*. Input to *cc* may consist of any of these types of files and translation may be stopped at any point.

Translation proceeds as follows:

a) Each *.c*, *.p* and *.f* input is run through the C macro preprocessor *cpp*. In the case of Pascal source, cpp is given the *–p* switch. This switch tells *cpp* to ignore Pascal-style comments and do the correct things with preprocessor control lines so that the line numbers in the resultant Pascal file will be the same as the original.

b) The preprocessed C files are then run through the C compiler *ccom* and, if specified, the C optimizer *c2*. The resulting *.s* files are then assembled, producing UNIX objects (*.o*).

c) Preprocessed FORTRAN (*.f*) and Pascal (*.p*) files are run through the appropriate SVS front end, *fortran* or *pascal*, then through the code generator *code*, producing special object files (*.j*). All special object files are combined with the FORTRAN/Pascal library and passed to an object file formatter *ulinker*, producing a single UNIX object file (*.o*).

d) Finally, all UNIX object files are passed to *ld(1)*, along with the UNIX startup file */lib/crt0.o*, to produce a single executable named *a.out*.

Preprocessed source files and assembler files are usually removed. All C objects (*.o*) and special object files (*.j*) are preserved, unless there was only a single input c file.

If C and FORTRAN files are mixed in a single executable, special interface routines must be generated as described in Appendix D of the *IRIS Workstation Guide*. If C and Pascal procedures are mixed, the user should consult

the SVS Pascal reference manual for instructions on altering the external procedure declarations in Pascal.

Options

The following options are interpreted by *cc* (*f77*,*p*c). Some options have meaning for only one of these languages, (see *ld*(1) for load-time options):

–c Suppress the loading phase of the compilation, and force an object file to be produced even if only one source file is given.

–g Generate debugging information. Currently, this does not have meaning when C is intermixed with another language. For FORTRAN and Pascal files, the appropriate compiler will be called with the +d switch and the symbol table produced by the pre–linker will be placed in *x*.dbg, where *x* is the name of the final program. For pure C programs, additional symbol table information will be generated for dbx(1).

–k Include lib*x*.a as a library *ld* should search for undefined references. *Ld* will look for the library first in the directory */lib*, then in */usr/lib*, and finally in */usr/local/lib* until it finds it. The string *x* may be more than one character.

––n Normally, *cc* passes the *–n* switch to *ld*, which causes it to load the program with *shared text*. The *––n* switch suppresses the passing of –n to *ld*.

–o output
Name the final output file *output* instead of *a.out*.

–p Tell *ccom* to generate code to count subroutine calls for use with *prof*. Neither FORTRAN nor Pascal support profiling.

––x By default, *cc* passes a *–x* flag to *ld*, in order to suppress local symbols from the final symbol table. The *––x* flag inhibits this default. Note that there are two dashes.

–C prevent the macro preprocessor from removing C style comments found in the source. Such comments are *always* removed from Pascal programs.

–Dname=def
–Dname
Define *name* to the preprocessor, as if by *#define*. If no definition is given, the name is defined as "1".

–E Run only the macro preprocessor on the named C, Pascal and FORTRAN source, and send the result to standard output.

–Idir Look in directory *dir* for missing *#include* files. Include files whose names are surrounded by double quotes and do not begin with '/' are always sought first in the directory of the input file, then in

directories named in *–l* options, then in */usr/include*, and finally in */usr/local/include*. Include files names beginning with '/' are treated as absolute paths. Include files whose names are surrounded by angle brackets (<) and (>) are not looked for in the directory of the input file.

–L Produce an assembly listing tor each C or assembler source file, and a FORTRAN listing of each FORTRAN source file. Assembler listings have the suffix *.lst* and FORTRAN listings have the suffix *.l*.

–Oxx Invoke an object–code optimizer on each C file. *xx* are optional flags to *c2*. Possible options are S (perform stack optimizations), P (remove stackprobes), K (omit kernel optimizations). Use of these options is not recommended for the standard compilation environment.

–P Run only the macro preprocessor on the named C, FORTRAN, and Pascal files, and place the results on *file.i*.

–S Compile the named files, leaving the C assembly language output in files suffixed *.s*, and the FORTRAN and Pascal objects in files suffixed *.j*.

–Uname
Remove any initial definition of *name*.

–Zf Cause instructions for the Sky floating point processor to be generated. When this switch is used, the Sky math library *–lmsky* will be substituted for the standard math library *–lm* if it is specified. Use of this switch on systems which do NOT have the floating point unit installed will cause a run time abort.

–Zg Load the program with the special files and libraries necessary for IRIS graphics programs. When this switch is used, the graphics library *–lg* and the math library *–lm* (or *–lmsky* if the *–Zf* flag has also been specified) are given by default. Special files must be loaded for using graphics with each source language. Hence, *cc* must be able to determine the combination of languages involved in the link step. If the compilation line specified *f77*, a FORTRAN source file (with the extension .f) or the switch –ZF is given, *cc* assumes that FORTRAN routines are present. In this case, the program is also loaded with the FORTRAN graphics interface library *–lfgl* and the FORTRAN object file containing the block data initialization of the common areas DEVICE and GL (*/usr/lib/fgldat.j*). If the compilation line specified pc, a Pascal source file (with the extension p) or the switch –ZP is given, *cc* assumes that Pascal routines are present. The program is loaded with the special Pascal jump table (*/usr/lib/pjmptbl.o*), and *ld* is told to make only eight characters significant in function names during calls to the graphics library.

–Zi filename
Use the file named *filename* as the run time startup, rather than the standard C run time startup. This is useful for generating standalone programs.

–Zq Time all subprocesses, and report these times on *stdout* at the end of the compilation.

–Zr Load the program for the *remote* graphics environment. If the source contains FORTRAN or *.j* files, the FORTRAN remote graphics library will be loaded, otherwise the C remote graphics library will be loaded. As does loading the standard graphics library, loading the remote graphics library automatically causes the math library (*–lm* or *–lmsky*) to be loaded. If the program is a C program, the directory */usr/include/rgl* will be searched prior to */usr/include* for graphics header files.

–Zv Turn on *verbose* mode. In verbose mode, the C compiler *ccom* will give additional diagnostics. This includes such things as flagging any use of the C type *double*, and complaining about too many register declarations.

–Zz Print a trace of all *exec()* calls.

–ZA pass the remainder of the string to *as*. Thus, the *cc* switch *–ZA–q* will pass as the switch *–q*.

–ZC pass the remainder of the string to *ccom*. Thus, the *cc* switch *–ZA–v* will pass ccom the switch *–v*.

–ZF pass the remainder of the string to the FORTRAN compiler front-end *fortran*. Thus, the *cc* switch *–ZF+d* will pass *fortran* the switch *+d*. This switch (with or without a switch to pass to the FORTRAN front–end) also informs *cc* that FORTRAN files were present in the compilation.

–ZM Cause the FORTRAN pre–linker to generate a load map of the FORTRAN program. This will be placed in a file by the same name as the executable file with the added extension *.fmap*.

–ZP Pascal files are present in this compilation, *cc* cannot determine this unless it sees a *.p* file or the name *pc* is used.

–ZZ Load the program for the standalone environment. This causes substitutions to be made for the C library and the C run time startup.

Other flags are passed to *ld*. The files may consist of any mix of C, object, FORTRAN, assembler, object or library files. The tiles are passed to *ld*, it opted, in the order given, to produce an executable program named *a.out* or that specified by the *–o* option.

FILES

file.c	C source file
file.f	FORTRAN source file
file.p	Pascal source file
file.j	Pascal and FORTRAN object files
file.o	object (relocatable) file
file.s	assembly file
a.out	executable file
/lib/ccom	C compiler
/lib/cpp	C preprocessor
/lib/crt0.o	run time startup
/lib/libc.a	C library
/usr/lib/paslib.obj	FORTRAN library
/usr/lib/fortran	FORTRAN front-end
/usr/lib/pascal	Pascal front-end
/usr/lib/code	FORTRAN code-generator
/usr/lib/ulinker	FORTRAN pre-linker
/bin/as	assembler
/bin/ld	linking loader
/usr/include	default include directory
/usr/lib/fgldat.j	block data routine for graphics commons
/usr/lib/pjmptbl.o	Pascal graphics jump table and C string converter

SEE ALSO

IRIS Workstation Guide Appendices D and E
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978
B. W. Kernighan, *Programming in C — a Tutoria*l
D. M. Ritchie, *C Reference Manual*
SVS FORTRAN Reference Manual
SVS Pascal Reference Manual
as(1), ccom(1), cpp(1), ld(1), extcentry(1), mkf2c(1), a.out(5)

BUGS

Two bugs are currently outstanding in the C compiler system. The first bug is the result of the compiler running out of temporary registers when compiling an expression as *junk[c].i += a* in a function which contains many register variables. The compiler aborts with the message *expression causes compiler loop, try simplifying*. Rather than to reserve an additional register for the compiler's internal use, it has been left to the user to alleviate this problem by reducing the number of data registers being used in this instance to five or less.

The second bug occurs when using the optimizer, *c2*. In certain situations, the optimizer gets confused as to the boundary between functions when optimizing functions which are declared as *static*. This can result in

optimizing away necessary register save/restore code at function entry and exit. Although the problem occurs rarely even in conjunction with such declarations, users are currently warned against declaring functions as *static* when using the optimizer.

DIAGNOSTICS

The diagnostics produced by C, FORTRAN, and Pascal are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

NAME

cube – real-time display of famous cube puzzle

SYNOPSIS

/usr/people/demos/cube

DESCRIPTION

Cube displays a moving, rotating, 3-D model of the well-known Rubik's cube puzzle. As the cube changes, the viewpoint translates in and out and moves around and around the object. Hidden surfaces are removed in real time. The mouse valuators and buttons control the display.

The motion will continue with the system unattended. This allows the program to be used as a stress test for the geometry system.

AUTOMATIC OPERATION

For automatic operation of the cube, use the menu at the top left. Move the cursor to the menu entry for the desired mode; press and release the right mouse button to select it. The *rotate* mode makes the cube alter itself while the viewpoint's altitude and azimuth change. *Translate* makes the viewpoint move back and forth. *Both* and *Freeze* do as one would expect.

MANUAL OPERATION

First select *Freeze*. The cube will come to rest as soon as its current internal motion is complete.

To rotate a surface of the cube, put the cursor on it. Tap the left mouse button to rotate the surface counterclockwise; the right one to turn it clockwise. If the side you wish to rotate is not visible, hold down the middle button and move the cursor. This changes viewing altitude and azimuth to bring the hidden surfaces into view.

The middle menu and the right button can tie the middle button to distance and field of view rather than altitude and azimuth.

Quit terminates the program. *Reset* initializes the program and, incidentally, solves the cube.

AUTHORS

Herb Kuta and Kurt Akeley

HARDWARE CONFIGURATION

Eight bitplanes of memory are necessary.

NAME

curve – fast interactive cubic curve display

SYNOPSIS

/usr/people/demos/curve

DESCRIPTION

Curve rapidly draws any of several cubic curves. All control uses the mouse and its buttons.

To change the display, press RIGHTMOUSE to display a popup menu. Move the cursor till the menu option you select is highlighted and release the button. To get rid of the menu without changing the display, move the cursor clear of the menu and release the button.

Add Point and *Delete Point* place basis points on the plane. To add one or more points, select *Add Point* , move the cursor to the point locations, and press LEFTMOUSE once for each new point. Each point will be labeled with a marker. As soon as four or more points are selected, they cubic curve they determine with the current basis is drawn. In *Delete Point* is selected, the nearest basis point to the cursor and the portion of the curve affected by that point are colored. LEFTMOUSE will delete the point.

In *Move Point* mode, the point nearest the cursor and the affected part of the curve are colored. LEFTMOUSE will move the point to the cursor. Hold LEFTMOUSE down to drag the curve.

Select *Motion* to give each point a random direction and velocity. Select *Freeze* to stop it. *Basis*, *Linestyle*, and *Precision* each present their own popup menus. Vary their parameters with RIGHTMOUSE. Return to the main menu from any of these with *Quit*.

Markers Off turns off the basis point markers. *Markers On* restores them. *Smear* simulates families of curves. *No Smear* restores the single curve. *3-D* and *2-D* select a 3-D box or the initial plane for the display.

Initialize reselects the initial state. *Quit* (from the main menu) terminates the program.

AUTHORS

Rocky Rhodes and Herb Kuta

BUGS

The Bezier curve looks wrong if more than four points are specified. It is not. It is discontinuous, but that's the way M. Bezier designed it.

HARDWARE REQUIREMENTS

Eight bitplanes and 1.5 Megabytes of memory are required to run *curve*.

NAME

dbx – debugger

SYNOPSIS

dbx [**–r**] [**–i**] [**–I** *dir*] [*objfile* [*coredump*]]

DESCRIPTION

Dbx is a tool for source level debugging and execution of programs under UNIX. The *objfile* is an object file produced by a compiler with the appropriate flag (usually "–g") specified to produce symbol information in the object file. On the IRIS workstation, only *cc*(1) produces the appropriate source information. The machine level facilities of *dbx* can be used on any program.

The object file contains a symbol table that includes the name of the all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named "core" exists in the current directory or a *coredump* file is specified, *dbx* can be used to examine the state of the program when it faulted.

If the file ".dbxinit" exists in the current directory then the debugger commands in it are executed. *Dbx* also checks for a ".dbxinit" in the user's home directory if there isn't one in the current directory.

The command line options and their meanings are:

–r Execute *objfile* immediately. If it terminates successfully *dbx* exits. Otherwise the reason for termination will be reported and the user offered the option of entering the debugger or letting the program fault. Dbx will read from "/dev/tty" when **–r** is specified and standard input is not a terminal.

–i Force *Dbx* to act as though standard input is a terminal.

–I *dir* Add dir to the list of directories that are searched when looking for a source file. Normally *dbx* looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.

Unless **–r** is specified, *dbx* just prompts and waits for a command.

Execution and Tracing Commands

run [*args*] [**<** *filename*] [**>** *filename*]
rerun [*args*] [**<** *filename*] [**>** *filename*]
Start executing *objfile*, passing *args* as command line arguments; **<** or **>** can be used to redirect input or output in the usual manner. When **rerun** is used without any arguments the previous argument list is passed to the program; otherwise it is identical to **run**. It *objfile* has been written since the last time the symbolic information was read in, *dbx* will read in the new information.

trace [**in** *procedure/function*] [**if** *condition*]
trace *source-line-number* [**if** *condition*]
trace *procedure/function* [**in** *procedure/function*] [**if** *condition*]
trace *expression* **at** *source-line-number* [**if** *condition*]
trace *variable* [**in** *procedure/function*] [**if** *condition*]
Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. Source line numbers in a file other than the current one must be preceded by the name of the file in quotes and a colon, e.g. "mumble.p":17.

If the argument is a procedure or function name then every time it is called, information is printed telling what routine called it, from what source line it was called, and what parameters were passed to it. In addition, its return is noted, and if it's a function then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause "**in** *procedure/function*" restricts tracing information to be

printed only while executing inside the given procedure or function.

Condition is a boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

stop if *condition*
stop at *source-line-number* [**if** *condition*]
stop in *procedure/function* [**if** *condition*]
stop *variable* [**if** *condition*]
Stop execution when the given line is reached, procedure or function called, variable changed, or condition true.

status [**>** *filename*]
Print out the currently active **trace** and **stop** commands.

delete *command-number ...*
The traces or stops corresponding to the given numbers are removed. The numbers associated with traces and stops are printed by the **status** command.

catch *number*
ignore *number*
Start or stop trapping signal *number* before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. Initially all signals are trapped except SIGCONT, SIGCHILD, SIGALRM and SIGKILL.

cont Continue execution from where it stopped. Execution cannot be continued if the process has "finished", that is, called the standard procedure "exit". *Dbx* does not allow the process to exit, thereby letting the user to examine the program state.

step Execute one source line.

next Execute up to the next source line. The difference between this and **step** is that if the line contains a call to a procedure or function the **step** command will stop at the beginning of that block, while the **next** command will not.

return [*procedure*]
Continue until a return to *procedure* is executed, or until the current procedure returns if none is specified.

Displaying and Naming Data

print *expression* [**,** *expression* ...]
Print out the values of the expressions. Array expressions are always subscripted by brackets ("[ ]"). Variables having the same identifier as one in the current block may be referenced as "*block-name* **.** *variable*". The field reference operator (".") can be used with pointers as well as records, making the C operator "–>" unnecessary (although it is supported). The construct *expression* \ *typename* can be used to print the *expression* out in the format of the type named *typename*.

whatis *name*
Print the declaration of the given name, which may be qualified with block names as above.

which *identifier*
Print the full qualification of the given identifier, i.e. the outer blocks that the identifier is associated with.

whereis *identifier*
Print the full qualification of all the symbols whose name matches the given identifier. The order in which the symbols are printed is not meaningful.

assign *variable* — *expression*
set *variable* = *expression*
Assign the value of the expression to the variable.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function. Currently, calls to a procedure with a variable number of arguments are not possible. Also, string parameters are not passed properly for C.

where Print out a list of the active procedures and function.

dump [**>** *filename*]
Print the names and values of all active variables.

up [*count*]
down [*count*]
Move the current function, which is used for resolving names, up or down the stack *count* levels. The default count is 1.

Accessing Source Files

edit [*filename*]
edit *procedure/function-name*
Invoke an editor on *filename* or the current source tile it none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable EDITOR to the name of the desired editor.

file [*filename*]
Change the current source file name to *filename*. If none is specified then the current source file name is printed.

func [*procedure/function*]
Change the current function. If none is specified then print the current function. Changing the current function implicitly changes the current source file to the one that contains the function; it also changes the current scope used for name resolution.

list [*source-line-number* [**,** *source-line-number*]]
list *procedure/function*
List the lines in the current source file from the first line number to the second inclusive. If no lines are specified, the next 10 lines are listed. If the name of a procedure or function is given lines *n–k* to *n+k* are listed where *n* is the first statement in the procedure or function and *k* is small.

use *directory-list*
Set the list of directories to be searched when looking for source files.

Machine Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [if *cond*]
stopi [*address*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]
Turn on tracing or set a stop using a machine instruction address.

stepi

nexti Single step as in step or next, but do a single instruction rather than source line.

address ,address / [*mode*]

[*address*] / [*count*] [*mode*]
Print the contents of memory starting at the first *address* and continuing up to the second *address* or until the items are printed. If no address is specified, the address following the one printed most recently is used. The *mode* specifies how memory is to be printed; if it is omitted the previous mode specified is used. The initial mode is "X". The following modes are supported:

i print the machine instruction
d print a short word in decimal
D print a long word in decimal
o print a short word in octal
O print a long word in octal
x print a short word in hexadecimal
X print a long word in hexadecimal
b print a byte in octal
c print a byte as a character
s print a string of characters terminated by a null byte
f print a single precision real number
g print a double precision real number

Symbolic addresses are specified by preceding the name with an "&". Registers are denoted by "$rN" where N is the number of the register. Addresses may be expressions made up of other addresses and the operators "+", "–", and indirection (unary "*").

Miscellaneous Commands

sh *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

alias *new-command-name old-command-name*
Respond to *new-command-name* as though it were *old-command-name*.

help Print out a synopsis of *dbx* commands.

gripe Invoke a mail program to send a message to the person in charge of *dbx*.

source *filename*
Read *dbx* commands from the given *filename*.

quit Exit *dbx*.

FILES

a.out	object file

.dbxinit initial commands

SEE ALSO

cc(1), f77(1)

BUGS

The version of dbx contained in release 2.1 is preliminary. It is a complex tool which has recently been ported to the IRIS workstation, and which will take some time to become stable. The major debugger commands work well on most programs. However, there are many minor difficulties which remain to be solved in this release. It is very important that difficulties which appear are documented with test cases and reported to Silicon Graphics to be addressed.

There are two major classes of bugs outstanding in dbx. The first class are bugs which are caused by limitations in the environment and which will probably not be fixed. The second are current bugs which remain to be fixed.

Inherent Bugs

dbx may not be able to trace back through the stack if a procedure has been executed which does not do a link (most assembler "routines, including those in the C library). It may, of course, not be able to display the parameters to a called procedure which has not been compiled with debugging information.

Non-local gotos can cause some trace/stops to be missed. Most of the command names are too long. The alias facility helps, but is really quite weak. A *csh*-like history capability would improve the situation. But then, who wants to duplicate the C-shell in a debugger?

Dbx suffers from the same "multiple include" malady that sdb does. If you have a program consisting of a number of object files and each is built from source files that include header files, the symbolic information for the header files is replicated in each object file. Since about one debugger start-up is done for each link, having the linker (Id) re-organize the symbol information won't save much time, though it would reduce some of the disk space used. The problem is an artifact of the unrestricted semantics of #include's in C; for example an include file can contain static declarations that are separate entities for each file in which they are included.

Outstanding Bugs

dbx in some cases touches illegal pages, causing a segmentation violation.

Certain circumstances cause self-diagnosed internal inconsistencies. These result in an 'internal error' message from dbx followed by an abort. Test cases which consistently cause these errors would be greatly appreciated.

dbx sets an incorrect breakpoint in certain cases when single stepping at the source level. This causes control of the program to be lost. In particular,

stepping through a case statement or the entry or exit of a loop seems to cause confusion.

dbx refuses to display ranges of either address or data registers, or to allow setting registers.

It is often very difficult to interrupt *dbx* when tracing. *Dbx* seems to field the SIGKILLs but not to respond to them. A stream of kill signals (holding down ^C) will force *dbx* to exit.

Tracing of items other than line numbers and global variables doesn't work correctly. Additionally, *dbx* seems to ignore breakpoints when it is tracing.

Dbx does not always set the breakpoint correctly when tracing the statement in a single-statement loop. If the single statement is surrounded by braces, tracing of the statement appears to work.

Besides the problems with not doing a link, floating point routines in the C library sometimes pass their arguments in registers. In this case, *dbx* will not see the arguments.

Other bugs in the system should be reported. In all cases, a bug report for *dbx* should include the smallest possible test case, the corresponding source, and an exact sequence of events which causes the failure.

NAME

dog – cooperative or competitive flight simulator using the ethernet

SYNOPSIS

/usr/demos/dog

DESCRIPTION

The *flight* demo is extended for multiple IRIS systems. Each IRIS determines the position of its aircraft several time a second, broadcasts the plane's location and orientation to the other IRIS systems running *dog* , and listens to the other planes' locations. All known planes in the current field of view are displayed on all systems.

Pilots may cooperate by attempting formation aerobatics or compete by trying to shoot each other down. The fighter are armed with missiles, rockets, and cannon. The coordinates of projectiles are included in the ethernet packages, hits are detected, and scoring is maintained.

WEAPONS

Fighters are armed with rockets, sidewinders, and cannon.

Rockets have about ten seconds of power and follow ballistic paths after the power is exhausted. They explode when they strike the ground, come within range of an aircraft (including the one they came from), or are destroyed by their owner.

Sidewinders are like rockets, but will steer themselves towards the nearest aircraft (except their owner's). Sidewinders can turn around and destroy aircraft behind you. The Cessna 150 does not generate enough heat to attract sidewinders. Sidewinders will not track aircraft on the ground. A good pilot can usually outmaneuver a sidewinder.

Cannon have limited range -- each shell exists for only one second.

The number of rockets and sidewinders available on each type of fighter are indicated on the help display. Landings replenish armament as well as fuel. The number of missiles replenished depends on the quality of the landing. Ammunition for the cannon is inexhaustible.

Each aircraft can have only one projectile in the air at a time. He can destroy projectiles that have missed their targets to allow new ones to be fired.

q fires a rocket, *w* fires a sidewinder, *e* fires the cannon, *r* destroys the current projectile. Any aircraft in the range of the explosion will be destroyed.

In *flight* , or in *dog* with no competition, strafing the airport can be good practice for the real thing.

SCORING

dog keeps track of kills and crashes. A pilot scores a kill when a projectile fired by his plane destroys another aircraft. A pilot scores a 'killed' when

his aircraft is destroyed by a projectile or crashes.

Each pilot's score is displayed on his instrument panel. The scores of all the current players are shown to each new player when he joins the game and when he reincarnates himself after destruction.

When a player joins the game, an announcement is broadcast to all players. Messages are sent whenever a player quit or is destroyed, as well as when sent by the *radar* program.

AUTHOR

Gary Tarolli

BUGS

See the bugs for *flight*.

Various kinds of cheating are possible. (For example, temporarily selecting the night display to better see a distant opponent's exhaust.) Some scrupulous pilots avoid operations not possible in real aircraft. Others use every trick possible.

The Cessna 150 and B-747 have 20mm cannon. This is inaccurate but amusing in the C-150 (the 747 is much too logy). Try taking off in the 150 and flying around the ramp, picking off opponents as they appear. Since the Cessna is invisible to sidewinders, it has a chance to survive. If you make a mistake and take off in the 747, you are dogmeat.

The cannon ammunition should be finite. The cannon, in fact, should overheat and jam if used too often.

NAME

flight – simulate the flight of any of several aircraft

SYNOPSIS

/usr/people/demos/flight

DESCRIPTION

One large viewport shows an instance of a world; several smaller ones simulate instruments. The world is viewed from the cockpit of an aircraft or from a control tower. The mouse and keyboard control the aircraft and its environment.

Starting Up

Flight provides two pages of help information. To freeze the action at any time and display the first page, *h*. Type any key to view the next page. Type any key to continue.

The first help page briefly describes the program. Read it, and press any key to continue. The second pages offers descriptions of five aircraft: one two-place trainer (Cessna 150), one heavy transport (Boeing 747), and three fighters. Type 1 to select the Cessna 150.

The view you see is from the cockpit of the Cessna. Type *d* to see the Cessna from the control tower. Type *x* a few times for a closer view. Type *d* to return to the cockpit and strike *s* three or four times to advance the throttle. The aircraft will start to taxi towards the runway. Type twice to raise the flaps — Cessnas normally take off that way. When the plane is almost on the runway, tap the right mouse button five or six times to apply right rudder. The plane will start to turn right. The left mouse button move the rudder one increment to the left; the center one sets the rudder to zero. Move the mouse till the cursor is centered on the bottom edge of the screen and tap *s* until the thrust indicator shows all blue. When the airspeed indicator passes 60 knots, move the mouse smoothly toward you. The cursor should be in the upper center of the attitude indicator. When the rate-of-climb indicator shows blue, you are flying! Congratulations!

Now turn around and land.

Flight Controls

Flight is controlled by the mouse, the mouse buttons, and the keyboard. The mouse holds the primary flight controls.

Rightmouse and *leftmouse* move the rudder one increment to the right and left respectively. *Middlemouse* centers it. The rudder position is shown by a small red triangle at the lower edge of the attitude indicator. The rudder is used primarily to maneuver the aircraft on the ground. Airborne turns are made, as in real aircraft, by coordinated application of aileron and elevator.

The mouse X and Y valuators control the ailerons and elevator, emulating a control stick. Left-right motion controls roll; forward-back motion controls

pitch. The stick position is indicated by a square white cursor. Both controls are at their neutral position when the cursor is centered at the bottom of the windshield. Stick position tor level flight is slightly below center.

The *s* key increases the throttle setting; the *a* key decreases it. The left bar indicator shows the throttle setting as a percentage of full power. Reverse thrust is available and shown in red. Thrust goes to zero when the plane climbs through 50,000 feet and the engine flames out. It can be restored by descending and applying throttle. Thrust goes to zero when fuel goes to zero. It can be restored only by making a safe landing (good luck) to pick up fresh fuel.

Secondary flight controls include the landing gear, flaps, and spoilers. To raise or lower the landing gear, type *l*. To increase or decrease the flaps, type *f* or *F*. To increase or decrease the spoilers, type *c* or *C*. Flap and spoiler ranges are determined by the aircraft. The Cessna has no spoilers and its gear is down and welded.

The landing gear has two functions: to protect the fuselage from the ground and to add drag. You may lower the gear to slow the plane down and make handling easier.

Flaps and gear are structurally unsound at high speeds. They fall off if you exceed 400 knots while they are deployed or if either is deployed at speeds over 400 knots. Missing flaps make good landings difficult. Missing gear makes a good landing impossible.

Flaps increase lift, increase drag, and decrease stall speed. Takeoffs are normally made with partial flaps; landings with full flaps.

Spoilers decrease lift and increase drag dramatically. They are most useful in dissipating excess altitude without increasing speed. It is difficult to recover from a stall while spoilers are deployed.

Display Controls

Several controls allow the viewer to alter his view of the world.

The *left-arrow* and *right-arrow* keys rotate the pilot's point of view 90 degrees to the left or right respectively. The viewing angle (front, left, rear, or right) is displayed on the windshield. The keys are useful for looking around, but remember to set the view back to the front for any but the simplest flying.

The *d* key switches the viewpoint from the cockpit to the control tower or back. The control tower always looks toward the plane, *x* decreases the tower's field of view, effectively magnifying the aircraft. *z* increases the field of view. If there is doubt as to whether the view observed is from the cockpit or the tower, observe the center of the window. A yellow tracking cross marks the cockpit view.

n changes the time of day from daylight to night or back. There is an interesting city NNW of the airport.

Instruments

This section lists the instruments on the panel from left to right.

The *thrust indicator* shows thrust as a percentage of full throttle. As with all the bar-graph displays, a blue bar is positive and a red one negative. Reverse thrust is possible only on the ground.

The *airspeed* indicator is calibrated from 0-1000 knots. (100 knots is about 118 miles per hour.) Negative airspeeds can happen during such acrobatic maneuvers as hammerhead stalls. Since wind is not simulated, airspeed ≡ groundspeed.

The *vertical-speed indicator* shows rate of climb in feet per minute. Note that the fighter (in normal operation) and the civil planes (usually while crashing) can exceed the 10,000 fpm maximum absolute rate displayed. Use the numeric display at the bottom of the band.

The *G-meter* indicates vertical acceleration. Each aircraft has maximum stress limits. I they are exceeded, the attitude indicator shows the message "G-LIMIT."

The *attitude indicator* or *artificial horizon* helps orient the plane when the real horizon is not visible. The triangular indicator at the bottom edge shows the rudder position.

The *fuel* gauge shows remaining fuel as a percentage of a full tank. To reduce fuel consumption to zero (for tests only) type ~. This is considered cheating in normal flight.

Landings, Crashes, and Restarts

A good landing is a landing on the runway, with gear down, a descent rate of less than 600 fpm, and in line with the runway. Good landings are rewarded with scores from 0 - 100 points. For every point scored, fuel on board is increased by 1% of total capacity. (Total capacity is never exceeded.)

Landings that are almost good (rate of descent, drift, and roll too high but not disastrous count as crash landings. You can keep flying, but get no more fuel.

Landings with the gear up, off the runway, or with excessive descent rate, drift, or roll count as "crashed into the swamps." All you can do is look at the wreckage from the tower or type *r* to restart the game.

r will not work while your plane is intact. Your plane must be destroyed. Your plane is destroyed if it crashes into the swamps, taxis too far off the runway, raises the gear while on the ground, or is shot down. If you wish to restart (to get a different kind of plane or refuel without landing), auger in or type *qr* to shoot yourself down. The latter will fail with unarmed civilian planes.

AUTHOR

Gary Tarolli

BUGS

Flight and its offspring are continually being improved. There is a significant creative spurt before each major trade show. Improvements may be documented in the program's help display before this page is affected.

The Cessna is much too difficult to bring out of a stall.

NAME

fload – initialize the Sky floating point processor

SYNOPSIS

/etc/fload

DESCRIPTION

Fload is run when the IRIS workstation is booted. It probes for the Sky-floating point processor. If the processor is found, it is initialized and its microcode loaded from the file */lib/skyffp.fas* .

The program can be run by the superuser under normal operating conditions. Since *fload* does not access the board through the normal kernel utilities, any users of the board may be aborted when the program is run.

DIAGNOSTICS

If *fload* is run by a user who is not the superuser, the message

fload: phys call failed

will be given.

If *fload* cannot access its microcode file (**/lib/skyffp.fas**), it will complain.

If the Sky board is not visible in the system, i.e., either it has not been installed or is installed incorrectly, the message

fload: skyffp not installed

will be displayed.

Fload does some cursory testing of the board. If one of these tests fails, one of the messages

fload: error initializing skyffp
fload: board test failure
fload: board function test failure

will be displayed. In any of these cases, the board should be replaced.

NAME

flow – display of complex scientific data base

SYNOPSIS

/usr/people/demos/flow

DESCRIPTION

Flow displays about 20,000 data points and a few vectors. The data is a three-dimensional display of the values of the Xavier-Stokes equations for turbulent fluid flow in a channel. The calculation was done at NASA-Ames on, of all things, the Illiac-IV.

Simple mouse motions control the display. Press LEFTMOUSE, move the cursor, and release the button to redraw the image with new altitude and azimuth. Use MIDDLEMOUSE to control distance and twist. RIGHTMOUSE starts a sequence of 100 short twists. Any keyboard key switches the display from single-buffer mode (the initial state) to double-buffer mode or back.

AUTHOR

Phil Gustafson.

HARDWARE CONFIGURATION

Eight bitplanes and 2.5 Megabytes of memory are required to run *flow*.

NAME

iib – initialize ib driver

SYNOPSIS

iib [-f ibtabfile] [-r] [-v]

DESCRIPTION

Iib initializes and starts the ib driver (see *ib*(8)). By default it uses information from the file */etc/ibtab*. The -f flag specifies an alternate file of the same format (see *ibtab*(5)). The -v flag causes *iib* to print each entry as it reads the file.

Normally *iib* has no effect if the ib driver is already started. The -r flag makes it re-initialize and re-start even if the driver was already started. The -r flag should only be used if the driver gets jammed, and only when no other programs are using the driver. Only the super-user may use the -r flag.

EXAMPLE

iib

FILES

/etc/ibtab

SEE ALSO

cib(8), dib(8), ibtab(3), ibtab(5), tib(8).

NAME

heme – depthcued display of the Cytochrome P450 protein molecule

SYNOPSIS

/usr/people/demos/heme

DESCRIPTION

Dr. Tom Poulos and Prof. Joseph Kraut of the University of California at San Diego have constructed a computer model of the Cytochrome P450 molecule to aid in their research into the structure of this enzyme. Computer graphics help in comparing electron density maps to the proposed protein structure by moving the hypothetical structure into regions of high electron density. Proximity of atoms provides information regarding chemical interaction.

Heme displays the Cytochrome protein with control of viewing transformations via pop-up menu selections. Also, display of various parts of the protein molecule can be toggled on and off.

Pop up the menu by holding down the CENTER mouse button. Menu selections are are toggled by placing the cursor atop the menu selection and pressing the LEFT mouse button. The menu disappears when the CENTER mouse button is released.

Left Menu	Function
5	Amino acids that interact with the heme group.
4	Backbone of P450, connecting alpha-carbons of amino acid subunits.
3	Electron density contour around the iron atom.
2	Van der Waal surface of the heme group.
1	The heme group, the subunit which accepts and donates electrons.

The depth cueing uses 36 intensities per color. The initial display is designed to emphasize the central structure of the model. If the model is translated far enough away to be completely visible, the depth cueing will render it virtually invisible. To see the whole model, scale it down (SCAL -) and translate it forward (TRAN +Z) until the perspective, Z-clipping, and depth cueing give an attractive image.

FILES

./hemlib/heme control program
./hemlib/heme[15] data files

HARDWARE CONFIGURATION

A minimum configuration of 16 bitplanes, 1.5 Megabytes of memory, and Z-clipping is required to run **heme**.

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [interval [count]]

DESCRIPTION

Iostat iteratively reports the number of characters read and written to terminals, and, for each disk, the number of seeks transfers per second, kilobytes transferred per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(1)

NAME

jet – depthcued wireframe model of the f18 jet fighter

SYNOPSIS

/usr/people/demos/jet

DESCRIPTION

A wireframe f18 model can be viewed under mouse control. The mouse buttons control rotation about the x, y & z axes and viewer distance. Unless otherwise noted, only the x-movement of the mouse is a significant valuator when holding down a mouse button.

Mouse buttons	Function
left	rotation about x-axis via mouse x-motion
middle	rotation about y-axis via mouse x-motion
right	rotation about z-axis via mouse x-motion
left & middle	translate object via mouse x & y motion
left & right	zoom back
middle & right	zoom forward
all	quit

Use the ENTER key to toggle into depth-cueing mode. The ARROW keys modify the portion of the color map that is used for the shaderange, effectively changing the intensity of the depthcue. Play with the arrow keys to get a more satisfying display.

The PF1 key toggles the cursor on & off.

HARDWARE CONFIGURATION

A minimum configuration of 8 bitplanes and 1.5 Megabytes of memory is required to run the jet demo.

NAME

ld – link editor

SYNOPSIS

ld [option] ... file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **–r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine (unless the **–e** option is specified).

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. The first member of a library should be a file named '__.SYMDEF', which is understood to be a dictionary for the library as produced by *ranlib*(1); the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for **–l**, they should appear before the file names.

–A This option specifies incremental loading, i.e. linking is to be done in a manner so that the resulting object may be read into an already executing program. The next argument is the name of a file whose symbol table will be taken as a basis on which to define additional symbols. Only newly linked material will be entered into the text and data portions of **a.out**, but the new symbol table will reflect every symbol defined before and after the incremental load. This argument must appear before any other object file in the argument list. The **–T** option may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding address (which must be a multiple of 1024). The default value is

the old value of _end.

–D Take the next argument as a hexadecimal number and pad the data segment with zero bytes to the indicated length.

–d Force definition of common storage even if the **–r** flag is present.

–e The following argument is taken to be the name of the entry point of the loaded program; location 0x1000 is the default. As UNIX will only currently recognize programs which begin execution at 0x1000 or 0x2000, the use of **–e** is limited to the standalone environment.

–l*x* This option is an abbreviation for the library name '/lib/lib*x*.a', where *x* is a string. If that does not exist, *ld* tries '/usr/lib/lib*x*.a' and '/usr/local/lib/lib*x*.a'. A library is searched when its name is encountered, so the placement of a **–l** is significant.

–M produce a primitive load map, listing the names of the files which will be loaded.

–N Do not make the text portion read only or sharable. (Use "magic number" 0407.)

–n Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read–only and shared among all users executing the file. This involves moving the data areas up to the first possible 1024 byte boundary following the end of the text.

–o The name argument after **–o** is used as the name of the *ld* output file, instead of **a.out**.

–r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

–S 'Strip' the output by removing all symbols except locals and globals.

–s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

–T The next argument is a hexadecimal number which sets the text segment origin. The default origin is 0x1000. (See **–e** notes.)

–t ("trace") Print the name of each file as it is processed.

–u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

–X Save local symbols except tor those whose names begin with 'L'. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines. NOTE: see the BUGS section.

–x Do not preserve local (non–.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

–y*sym* Indicate each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. (It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.)

–z Arrange for the process to be loaded on demand from the resulting executable file (413 format) rather than preloaded. This is the default. Results in a 1024 byte header on the output file followed by a text and data segment each of which have size a multiple of 1024 bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually appear (from the output of *size*(1)) to live in the data segment; this to avoid wasting the space resulting from data segment size roundup.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
/usr/local/lib/lib*.a	still more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), ranlib(1)

BUGS

There is no way to force data to be page aligned.

When the *–r* switch is used to preserve relocation bits, *ld* gets the relocation commands table out of sync with the symbol table if *–x* is given without *–X*. Thus one should not use *–X* with *–r*.

NAME

library – create a library of FORTRAN and Pascal objects

SYNOPSIS

library –l ofile ifile [ifile ....]
library listfname ofile ifile [ifile ....]

DESCRIPTION

Library can be used to generate a library of FORTRAN and Pascal object (.j) files. The resultant library, which should be suffixed with .j, can be used as as regular object to the *f77(1)* or *pc(1)* program. The prelinker program *code*, will recognize the library format and only extract needed objects.

Library accepts as its first argument either a filename, on which a listing of the created library is placed, or the switch *-l* , which is used to indicate that no library listing is desired. Library can also be run interactively, in which case it prompts tor the file names.

If the resultant library file is not suffixed with *.j*, *f77(1)* and *pc(1)* will not recognize it as a FORTRAN/Pascal file.

NAME

mkboot – make a "bootable" tape for system restoration

SYNOPSIS

/etc/mkboot [**–s** *standalonedir*] [**–r** *rootdev*] [*filesys*] ...

DESCRIPTION

Mkboot creates a tape with the contents of the disk at the time it is invoked. It is suitable for use in restoring a system to a functional state, regardless of the contents of the disk. Such a tape can be used to recover from a crash, or to bring up a new disk.

The tape contains two or more files: the first is a cpio-format archive containing necessary standalone programs; the second is a "dd" image of the root file system; the third and succeeding files are cpio-format archives of user file systems. The standalone programs are normally taken from /stand; the **–s** option may be used to specify an alternate directory. The root file system is normally **md0a** or **ip0a**, depending on the system model; the **–r** option may be used to specify an alternate (block) device. The remaining options are the path names of directories to be included on the tape; no user file systems are included by default.

To use the tape to restore a system, the standalone program "fex" (or "ipfex") is booted from the first file on the tape, then used to copy the second file (the root file system) onto the disk. All previous contents on the root file system of the disk are lost. The system may then be brought up in single-user mode. To restore user partitions, "newfs" may be used to create the user file system if necessary, "smt" is used to partition the tape to the third file, and then "cpio" may be used to read in the user files.

EXAMPLE

```
mkboot /usr
```

This command creates a standard "mkboot" tape.

SEE ALSO

cpio(1), smt(1).

Fex Formatter and Exerciser

Ipfex Interphase Formatter and Exerciser

NAME

newfs – create a new file system

SYNOPSIS

newfs fs

DESCRIPTION

Newfs takes a file system descriptor and generates a *mkfs* with the proper arguments. E.g.:

```
newfs md1c
newfs ip0d
```

Prior to performing the mkfs, the user is prompted with the mkfs line to be executed, and a 't' is required to perform the mkfs.

FILES

/dev/rfs

SEE ALSO

mkfs(1), sgilabel(1)

NAME

patran – PATRAN simulation of the space shuttle

SYNOPSIS

/usr/people/demos/patran

DESCRIPTION

A PATRAN shuttle model may be viewed under mouse control. The mouse buttons control rotation about the x, y and z axes, viewer distance, and z-clipping. Unless otherwise noted, only the x-movement of the mouse is a significant valuator when holding down a mouse button. Any key toggles opening & closing motion of the bay doors on the shuttle.

Commands	
Mouse button	*Function*
Left	Rotation about x-axis with mouse x-motion
Middle	Rotation about y-axis with mouse x-motion
Right	Rotation about z-axis with mouse x-motion
Left & middle	Translate object with mouse x & y motion
Left & right	Zoom back
Middle & right	Zoom forward
All	Move near z-clipping plane with mouse x-motion

HARDWARE REQUIREMENTS

A minimum configuration of 8 bitplanes and 1.5 Megabytes of memory is required to run the shuttle.

NAME

ps – process status

SYNOPSIS

ps [**acegklstuvwx**#]

DESCRIPTION

Ps prints information about processes. Normally, only your processes are candidates to be printed by *ps*; specifying **a** causes other users processes to be candidates to be printed; specifying **x** includes processes without control terminals in the candidate pool.

All output formats include, for each process, the process id PID, control terminal of the process TT, cpu time used by the process TIME (this includes both user and system time), the state STAT of the process, and an indication of the COMMAND which is running. The state is given by a sequence of four letters, e.g. "RWNA". The first letter indicates the runnability of the process: R for runnable processes, T for stopped processes, P for processes in page wait, D for those in disk (or other short term) waits, S for those sleeping for less than about 20 seconds, and I for idle (sleeping longer than about 20 seconds) processes. The second letter indicates whether a process is swapped out, showing W if it is, or a blank if it is loaded (in-core). The third letter indicates whether a process is running with altered CPU scheduling priority (nice); if the process priority is reduced, an N is shown, if the process priority has been artificially raised then a '<' is shown; processes running without special treatment have just a blank.

The options are:

a asks for information about all processes with terminals (ordinarily only one's own processes are displayed).

c prints the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.

e asks for the environment to be printed as well as the arguments to the command.

g asks for all processes. Without this option, *ps* only prints "interesting" processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.

k causes the file */vmcore* is used in place of */dev/kmem* and */dev/mem*. This is used for postmortem system debugging.

l asks for a long listing, with fields PPID, CP, PRI, XI, ADDR, SIZE, RSS and WCHAN as described below.

s adds the size SSIZ of the kernel stack of each process (for use by system maintainers) to the basic output format.

t*x* restricts output to processes whose controlling tty is *x* (which should be specified as printed by *ps*, e.g. *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no tty, *t* for processes at the current tty, etc). This option must be the last one given.

u produces a user oriented output. This includes fields USER, %CPU, NICE, SIZE, and RSS as described below.

v causes a version of the output which contains virtual memory statistics. This includes fields RE, SL, PAGEIN, SIZE, RSS, TSIZ, TRS, %CPU and %MEM, described below.

w uses a wide output format (132 columns rather than 80); if repeated, e.g. ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

x even supplies information about processes with no terminal.

specifies a process number, (indicated here by #), in which case the output is restricted to that process. This option must be last.

A second argument is taken to be the file containing the system's namelist. Otherwise, /vmunix is used. A third argument tells *ps* where to look for *core* if the **k** option is given, instead of /vmcore. If a fourth argument is given, it is taken to be the name of a swap file to use instead of the default /dev/drum.

Fields which are not common to all output formats:

USER	name of the owner of the process
%CPU	cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.
NICE	(or NI) process scheduling increment
SIZE	virtual size of the process (in 1024 byte units)
RSS	real memory (resident set) size of the process (in 1024 byte units)
TSIZ	size of text (shared program) image
TRS	size of resident (real memory) set of text
%MEM	percentage of real memory used by this process.
RE	residency time of the process (seconds in core)
SL	sleep time of the process (seconds blocked)
PAGEIN	number of disk i/o's resulting from references by the process to pages not loaded in core.
UID	numerical user-id of process owner

PPID numerical id of parent of process
CP short-term cpu utilization factor (used in scheduling)
PRI process priority (non-positive when in non-interruptible wait)
ADDR swap address of the process
WCHAN event on which process is waiting (an address in the system).

F flags associated with process as in *<sys/proc.h>*:

SLOAD	000001	in core
SSYS	000002	swapper or pager process
SLOCK	000004	process being swapped out
SSWAP	000008	save area flag
STRC	000010	process is being traced
SWTED	000020	another tracing flag
SULOCK	000040	user settable lock in core
SPAGE	000080	process in page wait state
SKEEP	000100	another flag to prevent swap out
SWEXIT	000200	working on exiting
SPHYSIO	000400	doing physical i/o (bio.c)
STIMO	000800	timing out during sleep
SGR	001000	process using graphics
SPTECHG	002000	ptes for process have changed
SPHYS	004000	process has phys region

A process that has exited and has a parent, but has not yet been waited for by the parent is marked < defunct >; a process which is blocked trying to exit is marked < exiting >; *Ps* makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory
/dev/drum	swap device
/vmcore	core file
/dev	searched to find swap device and tty names

SEE ALSO

kill(1), w(1)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation.

NAME

reshape – reshape the console textport

SYNOPSIS

reshape -s
reshape -40
reshape -24
reshape *llx lly xlen ylen*

DESCRIPTION

Reshape allows the textport to be reshaped. The textport is the window that is used for non-graphics programs. Some programs, such as *vi*, assume that the window is a particular size according to the specified termcap entry. The entry **wsiris**, for example, assumes a window of 40 lines by 80 columns. One may add entries to **/etc/termcap** to support different window sizes.

The **–s** flag causes reshape to set the window to be 40 rows by 80 columns in the center of the screen. The **–40** flag is equivalent to the **–s** flag. The **–24** flag causes *reshape* to set the window to be 24 rows of 80 columns in the center of the screen.

Instead of selecting a standard configuration, the size and location may be set to any desired values. The x and y values of the lower left corner of the textport window may be specified with *llx* and *lly*, in screen coordinates (pixels). The size of the window may be specified with *xlen* and *ylen* which specify the number of columns and rows respectively, in characters.

SEE ALSO

tpblank(1), tpon(1)

NAME

robot

SYNOPSIS

/usr/people/demos/robot

DESCRIPTION

Robot displays a model of an imaginary robot arm. Operator controls adjust the viewing parameters, the articulation of the arm, and the illumination.

Most user interface is through the mouse and its buttons. Press some combination of buttons and move the mouse to alter the display. The functions selected are displayed on the lower edge of the screen.

In the default case, the simulated object is lit as by a point source behind the viewpoint. Type *S* to simulate light from an arbitrary distant point source. Hold down MIDDLEMOUSE and RIGHTMOUSE and move the mouse to alter the altitude and azimuth of the light source. Other combinations of buttons still allow viewpoint modification and articulation of the object. Type *s* to return to normal illumination.

AUTHOR.

Kurt Akeley.

BUGS

The hidden-surface removal fails at a few extreme positions of the object.

HARDWARE CONFIGURATION

A minimum configuration of 12 bitplanes is required.

NAME

sgilabel – print the SGI disk label from a drive

SYNOPSIS

sgilabel drive
or
sgilabel fs

DESCRIPTION

Sgilabel has two forms. If a drive is specified (e.g., md0, ip1), *sgilabel* will print out the whole drive label, as defined in the *fex* documentation. This information includes the drive label, serial number, type of drive and controller, size (cylinders/heads/sectors), and the sizes of each file system on the drive.

If a fs (file system) is specified (e.g., md0a, ip1d), *sgilabel* will print out the size of the file system and the size of a cylinder (both in 512 byte blocks). (*Newfs*(1) uses this information.)

FILES

/dev/md*h	The *h* partition accesses the label
/dev/ip*h	The *h* partition accesses the label

SEE ALSO

newfs(1)

NAME

shuttle

SYNOPSIS

/usr/people/demos/shuttle

DESCRIPTION

A PATRAN shuttle model can be viewed under mouse control. The mouse buttons control rotation about the x, y and z axes, viewer distance, and z-clipping. Unless otherwise noted, only the x-movement of the mouse is a significant valuator when holding down a mouse button. Any key toggles opening & closing motion of the bay doors on the shuttle.

Mouse buttons	Function
left	rotation about x-axis via mouse x-motion
middle	rotation about y-axis via mouse x-motion
right	rotation about z-axis via mouse x-motion
left & middle	translate object via mouse x & y motion
left & right	zoom back
middle & right	zoom forward
all	move near z-clipping plane via mouse x-motion

HARDWARE CONFIGURATION

A minimum configuration of 8 bitplanes is required to run the shuttle.

NAME

smt – streaming magnetic tape manipulation program

SYNOPSIS

smt [**–t** */dev/tapename*] *command* [*count*]

DESCRIPTION

Smt is used to give commands to a Quarter Inch streaming magnetic tape drive. If a tape name is not specified, the default tape drive is used. *Smt* uses the default tape device **/dev/rmtioctl**. By default *smt* performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The tape default device has the ioctl minor to facilitate the use of opening and reading the tape when either using a no rewind device or a standard rewind and write file mark on close of the tape device.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

eof Write *count* end-of-file marks at the current position on the tape.

fsf Forward space *count* files.

fsr Forward space *count* records.

rewind Rewind the tape (*Count* is ignored.)

status Print status information about the tape unit. (*Count* is ignored.)

help Print command usage information about the command. (*Count* is ignored.)

Smt returns a 0 exit status when the operation(s) were successful, *smt* will return a 1 if the command was unrecognized, and 2 if an operation failed. *Smt* without any arguments will print the help command.

FILES

/dev/rqic	Raw magnetic Quarter Inch Cartridge Tape drive
/dev/nrqic	No rewind Quarter Inch Cartridge Tape drive
/dev/nrmt*	No rewind Quarter Inch Cartridge Tape drive
/dev/rmtioctl	Default Raw magnetic Quarter Inch Cartridge Tape drive

BUGS

Smt will sleep when accessing the tape if tape is busy and will awaken only after the tape is closed from a previous operation.

SEE ALSO

smtio(4)

NAME

tpblank – prohibits the system from updating the textport

SYNOPSIS

tpblank

DESCRIPTION

Tpblank prohibits the textport from being updated by non-graphics programs or when graphics programs exit. This is useful for preventing UNIX from affecting the screen image while running a series of graphics programs, each of which produces part of a complete image.

SEE ALSO

reshape(1), tpon(1)

NAME

tpon – turn on the console textport

SYNOPSIS

tpon

DESCRIPTION

Tpon allows the textport to be updated by non-graphics programs and when graphics programs exit. It is useful for re-enabling the textport after invoking *tpblank*.

SEE ALSO

reshape(1), tpblank(1)

NAME

uptime – show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

FILES

/vmunix system name list

SEE ALSO

w(1)

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–fs**] [interval [count]]

DESCRIPTION

Vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and cpu activity. If given a **–f** argument, it instead reports on the number of *forks* since system startup and the number of pages of virtual memory involved forking. If given a **–s** argument, it instead prints the contents of the *sum* structure, giving the total number of several kinds of paging related events which have occurred since boot.

If none of these options are given, *vmstat* will report in the first line a summary of the virtual memory activity since the system has been booted. If *interval* is specified, then successive lines are summaries over the last *interval* seconds, "vmstat 5" will print what the system is doing every five seconds; this is a good choice of printing interval since this is how often some of the statistics are sampled in the system; others vary every second, running the output for a while will make it apparent which are recomputed every second. If a *count* is given, the statistics are repeated *count* times. The format fields are:

Procs: information about numbers of processes in various states,

r	in run queue
b	blocked for resources (i/o, paging, etc.)
w	runnable or short sleeper (< 20 secs) but swapped

Memory: information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes which are running or have run in the last 20 seconds. A "page" here is 4096 bytes.

avm	active virtual pages
fre	size of the free list

Page: information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

re	Page reclaims
at	text pages lost between execs
pi	pages paged in
po	pages paged out
fr	pages freed per second
de	anticipated short term memory shortfall
sr	pages scanned by clock algorithm, per-second

up/hp/rk: Disk operations per second (this field is system dependent).

Typically paging will be split across several of the available drives. The number under each of these is the unit number.

Faults: trap/interrupt rate averages per second over last 5 seconds.

in	(non clock) device interrupts per second
sy	system calls per second
cs	cpu context switch rate (switches/sec)

Cpu: breakdown of percentage usage of CPU time

us	user time for normal and low priority processes
sy	system time
id	cpu idle

FILES

/dev/kmem, /vmunix

AUTHORS

William Joy and Ozalp Babaoglu

BUGS

There should be a screen oriented program which combines *vmstat* and *ps*(1) in real time as well as reporting on other system activity.

NAME

w – who is on and what they are doing

SYNOPSIS

w [**–h**] [**–s**] [user]

DESCRIPTION

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the user's login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

The **–h** flag suppresses the heading. The **–s** flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. **–l** gives the long output, which is the default.

If a *user* name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), ps(1)

AUTHOR

Mark Horton

BUGS

The notion of the "current process" is muddy. The current algorithm is "the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal". This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, *w* prints "-".)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or arguments which are garbage. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

NAME

wsiris – emulate an IRIS terminal with a workstation

SYNOPSIS

wsiris [**–hnrs**] hostname

DESCRIPTION

Wsiris allows a Workstation to emulate an IRIS terminal communicating over **XNS**. The actual connection (as opposed to the one emulated) may be either via **XNS** or over a serial line running at 1200 or 9600 baud.

When connecting via **XNS**, hostname should be the name of the system to connect to (set by **hostname** prior to invoking **xnsd** on the host system). When connecting via a serial line, hostname should be the word "serial".

The options are:

–h Half duplex line.

–n The terminal will not send X-on/X-off commands to the host.

–r Convert NL to CR for VMS connections.

–s Slow speed (1200 baud). Usable on serial connections only.

The serial connection uses port 3, which is /dev/ttyd2. To exit *wsiris*, type:

~.

NAME

ibtab – package for dealing with ibtab files

SYNOPSIS

setibfile(file) char *file;

int setibent()

endibent()

struct ibtab *getibent()

int ibnflags(flagstr,_val) char *flagstr; int *_val;

DESCRIPTION

This is a basic set of routines for dealing with *ibtab*(5) format files such as */etc/ibtab*. By default */etc/ibtab* is used. *Setibfile* changes the default within the calling program.

Setibent opens and positions the file at the origin. *Getibent* returns a pointer to a (static) structure containing the "next" entry. 0 is returned at the end-of-file. The structure is declared in the include file *<ibtab.h>*. *Endibent* closes the file.

Ibnflags interprets the Flags (.ibt_flags) field, saving its numeric equivalent in the caller-supplied _val. The return value is 0 if the string is a legitimate expression, -1 otherwise.

EXAMPLE

```
#include "ibtab.h"
extern struct ibtab *getibent();
struct ibtab *ip;
int val;
...
setibent();
while( (ip = getibent()) != 0 )
{
   if( ibnflags(ip-> ibt_flags, &val) < 0 )
         printf("illegal flags string %s\n",ip->ibt_flags);
   else
         printf("%s --> 0x%x\n",ip->ibt flags,val);
}
endibent();
```

FILES

/etc/ibtab

SEE ALSO

ib(4), iib(1), ibtab(5)

BUGS

Getibent stores data in static areas. Return values are over-written by subsequent calls.

NAME

a.out – assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

A.out is the output file of the assembler *as*(1) and the link editor *ld*(1). Both programs make *a.out* executable if there were no errors and no unresolved external references. Layout information as given in the include file is:

```
/*
 * Header prepended to each a.out file.
 */
struct exec {
        long a_magic;       /* magic number */
        unsigned a_text;    /* size of text segment */
        unsigned a_data;    /* size of initialized data */
        unsigned a_bss;     /* size of uninitialized data */
        unsigned a_syms;    /* size of symbol table */
        unsigned a_trsize;  /* size of text relocation */
        unsigned a_drsize;  /* size of data relocation */
        unsigned a_entry;   /* entry point */
};

#define OMAGIC 040          /* old impure format */
#define NMAGIC 0410         /* read-only text */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offsets to
 * text | symbols | strings.
 */
#define N_BADMAG(x) \
    (((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGIC )

#define N_TXTOFF(x) \
        (sizeof (struct exec))
#define N_SYMOFF(x) \
        (N_TXTOFF(x) + (x).a_text+(x).a_data \
        + (x).a_trsize+(x).a_drsize)
#define N_STROFF(x) \
        (N_SYMOFF(x) + (x).a_syms)
```

The file has five sections: a header, the program text and data, relocation information, a symbol table and a string table (in that order). The last three may be omitted if the program was loaded with the '–s' option of *ld* or if the

symbols and relocation have been removed by *strip*(1).

In the header the sizes of each section are given in bytes. The size of the header is not included in any of the other sizes.

When an *a.out* file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0x1000 in the core image; the header is not loaded. If the magic number in the header is OMAGIC (0407), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. This is the oldest kind of executable program and is rarely used. If the magic number is NMAGIC (0410) the data segment begins at the first 4096 byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. This is the default format produced by *ld*(1).

After the header in the file follow the text, data, text relocation data relocation, symbol table and string table in that order. The text begins immediately following the header, as given by the N_TXTOFF macro. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The symbol table follows all this; its position is computed by the N_SYMOFF macro. Finally, the string table immediately follows the symbol table at a position which can be gotten easily using N_STROFF. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table; this size INCLUDES the 4 bytes, the minimum string table size is thus 4.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
/*
 * Format of a symbol table entry.
 */
struct nlist {
        union {
            char        *n_name;   /* for use when in-core */
            long        n_strx;    /* index into file string table */
        } n_un;
        unsigned char   n_type;    /* type flag, i.e. N_TEXT etc */
        char            n_other;
        short           n_desc;    /* see <stab.h> */
        unsigned        n_value;   /* value of this symbol */
};
#define   n_hash        n_desc     /* used internally by ld */
```

```
/*
 * Simple values for n_type.
 */
#define   N_UNDF      0x0       /* undefined */
#define   N_ABS       0x2       /* absolute */
#define   N_TEXT      0x4       /* text */
#define   N_DATA      0x6       /* data */
#define   N_BSS       0x8       /* bss */
#define   N_COMM      0x12      /* common (internal to ld) */
#define   N_FN        0x1f      /* file name symbol */

#define   N_EXT       01        /* external bit, or'ed in */
#define   N_TYPE      0x1e      /* mask tor all the type bits */

/*
 * Other permanent symbol table entries
 * have some of the N_STAB bits set.
 * These are given in <stab.h>
/*
#define   N_STAB      0xe0      /* if any of these set, keep /*

/*
 * Format for namelist values.
/*
#define   N_FORMAT "%08x"
```

In the *a.out* file a symbol's n_un.n_strx field gives an index into the string table. A n_strx value of 0 indicates that no name is associated with a particular symbol table entry. The field n_un.n_name can be used to refer to the symbol name only if the program sets this up using n_strx and appropriate data from the string table.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader *ld* as the name of a common region whose size is indicated by the value of the symbol.

The value of a byte in the text or data which is not a portion of a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the bytes in the file.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

```
/*
 * Format of a relocation datum.
 */
struct relocation_info {
        int         r_address;          /* address which is relocated */
        unsigned    r_symbolnum:24,     /* local symbol ordinal */
                    r_pcrel:1,          /* was relocated pc relative */
                    r_length:2,         /* 0=byte, 1=word, 2=long */
                    r_extern:1,         /* doesn't include sym's value */
                    :4;                 /* nothing, yet */
};
```

There is no relocation information if a_trsize+a_drsize==0. If r_extern is 0, then r_symbolnum is actually a n_type for the relocation (i.e. N_TEXT meaning relative to segment text origin.)

SEE ALSO

as(1), ld(1), strip(1), nm(1), adb(1), dbx(1), stab(4)

NAME

ib – IEEE 488 interface

SYNOPSIS

device ib0 at mb0 csr 0x020 priority 5 vector ibintr
device ib1 at mb0 csr 0x100 priority 5 vector ibintr

DESCRIPTION

This is the driver for the National Instruments GPIB-796 IEEE 488 bus controller. Talker and Listener functions are provided. The driver may be initialized to provide Controller functions as well. After initialization and startup (eg, with *iib*(1l)), bytes written to the device file are talked to the corresponding listener, and bytes read from the device file are listened from the corresponding talker. The "End" bit is sent with the last byte of each write request. Read requests return up to the next "End" byte. Writes are synchronous.

From the driver's perspective, the 488 bus connects up to 8 "nodes" (physical slots), of which its controller is the 0th. The low-order 3 bits of the minor device number are the correspondent's node number (this is not the same thing as its gpib address). The remaining bits specify which controller if the workstation has more than one.

The driver must be fully initialized before it can be used for file i/o, since it needs a certain amount of "system integrator" information to function. In particular, the driver will function as Controller In Charge if the initialization (see *IBIOSETNODE*) designates it as System Controller. Startup marks the end of initialization. Initialization and miscellaneous functions are provided via *ioctl*(2). Some of these functions are available only to the super-user. The codes and structures are defined in the standard include file <*sys/lib_ioctl.h*>.

ioctl(fd,IBIOGETNODE,struct_sgnode_ptr)
: Fills in the *struct_sgnode_ptr->node* field with the driver's current idea of the per-node system integrator information for the node (physical slot) number given in the *struct_sgnode_ptr->slotno* field.

ioctl(fd,IBIOSETNODE,struct_sgnode_ptr)
: Changes the driver's per-node system integrator information for the node number given in the *struct_sgnode_ptr->slotno* field, according to the *struct_sgnode_ptr->node* field. Super-user only, and only before startup.

ioctl(fd,IBIOSTART,int_ptr)
: Ends initialization and allows normal file i/o to occur if **int_ptr* is non0, otherwise just re-enters initialization. Super-user only.

ioctl(fd,IBIOINIT,0)
: Initializes (or re-initializes) the driver's internal data structures. This

call should only be made when no other processes are using the driver. Super-user only.

ioctl(fd,IBIOPPC,int_ptr)
Remotely configures the given nodes for parallel polling. If bit *i* of **int_ptr* is 1, node *i* is configured according to its *n_ppr* as set by the IBIOSETNODE ioctl. Only nodes with *IBN_VALID* and *IBN_PPC* set in the *n_flags* field are configured. By default, node *i* uses line *i* for parallel polling. This call is not interruptible.

ioctl(fd,IBIOPOLL,char_15_ptr)
Fills in the given array with the responses from the last poll; position i is the response from node *i*. Responses with the 0100 (octal) bit turned on are considered active; other responses are considered inactive and are ignored. If no poll has been done since the previous IBIOPOLL call, return is not until the next poll occurs. The driver performs polls automatically in response to Service Request events: a parallel poll is done first, followed by a serial poll of all matching nodes known to be capable of responding to a parallel poll, plus all nodes known to be incapable of responding to a parallel poll. Parallel poll responses are assumed to be "in-phase," ie, logic 1 when active. This call is effective only if issued from the Controller In Charge side.

ioctl(fd,IBIOSRQ,int_ptr)
Raises Service Request with **int_ptr* as poll status. The poll status must have the 0100 (octal) bit turned on. Returns when serial-polled or timeout (currently about 10 seconds). This call is not interruptible.

ioctl(fd,IBIOTAKECTL,0)
Waits for the interface to become Controller In Charge. Super-user only, and only after startup. This cannot take effect until the driver is quiescent. This call is not interruptible.

ioctl(fd,IBIOPASSCTL,int_ptr)
Passes control to the node indicated by **int_ptr*. Super-user only, and only after startup. This cannot take effect until the driver is quiescent. This call is not interruptible. This call is effective only if issued from the Controller In Charge side.

ioctl(fd,IBIOCUTOFF,0)
Unaddresses the current talker and listener. This does not interfere with normal driver operation (normally the driver does not unaddress the current talker or listener until another talker or listener is selected; this call unaddresses and forces re-addressing next time). This call is effective only if issued from the Controller In Charge side.

In general *iib*(1l) is sufficient for dealing with ib driver ioctl's.

The most efficient blocksize for writes is 1024 bytes. Throughput may reach 110 *1024 bytes per second. The IEEE 488 "compliance" code for this device and driver is approximately

SH1 AH1 C5 T8 L4 SR1 RL0 PP1

FILES

/dev/ib[01][0-7]

SEE ALSO

cib(8l), dib(8l), iib(1l).

DIAGNOSTICS

See *intro*(2).

BUGS

When the interface is not Controller In Charge, the driver has no knowledge of the actual destination when it talks or the actual source when it listens: data from write requests is talked to whatever node happens to be listening when the interface is addressed to talk; listened data is put in an anonymous queue, which is used to satisfy read requests as if it had come from the requested node. In practical terms, this means that when the interface is not Controller In Charge it only talks or listens to the actual Controller In Charge.

Due to hardware limitations, killing a program while it's listening may occasionally scramble subsequent transfers.

Due to hardware limitations, the driver may occasionally hang when addressed to listen.

Responds to Service Requests only when the file is open.

No raw i/o. No asynchronous i/o.

NAME

smtio – UNIX streaming magnetic tape interface

DESCRIPTION

The special file */dev/rmt1* refers to the UNIX streaming magtape drive, which is on the MULTIBUS using the DSD-5217 controller. The following description applies to any of the transport/controller pairs. The special files */dev/rmt1*, */dev/qic*, and */dev/mt1* are 10000 fci, 450 ft, 45 ips, 45 MByte Quarter Inch Tape streaming drives, e.g., Archive, Wangtek or Cipher. */dev/nrqic*, */dev/nrmt1*, and */dev/nmt1* are no rewind devices with the same specifications as above. */dev/nrmt1* is the special file meant as the default to *smt* commands. Refer to *smt*(1) for the specifications of ioctl commands to manipulate the tape drives. The files */dev/rqic*, */dev/rmt1*, and */dev/mt1* are rewound when closed; the others are not. These files will also be closed by writing a file mark. The other files will not be rewound upon close. They will also write a file mark but will be positioned at the file mark for additional files to be added to the tape cartridge.

A standard tape consists of a series of 512 byte records terminated by an end-of-file. The system makes it possible to treat the tape like any other file. Seeks do not have their usual meaning and it is not possible to read or write a byte at a time. Writing in very small blocks (less than 5120 bytes) is inadvisable because this tends to create large record gaps and causes the tape to stop streaming. The tape drive must then reposition the tape cartridge for the next write or read. This causes a slower performance due to the tape moving backwards and forwards and stopping and starting.

The *smt*(1) manipulation program discussed above is useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the raw' interface is appropriate. The standard format for referring to the 'blocked' device is */dev/mt1*, but the 'raw' and the 'blocked' devices are the same for the Quarter Inch Streaming tape drive. The associated files are named */dev/rmt1* and */dev/rqic* but the same minor device considerations as for the regular devices still apply. A number of ioctl operations are available on raw magnetic tape. Refer to *smt*(1) for additional information for use with */dev/nrmt1*. The following definitions are from */usr/include/sys/mtio.h*:

```
/*
 * Structures and definitions for mag tape io control commands
 */

/* structure for MTIOCTOP - mag tape op command */
struct    mtop      {
          short     mt_op;          /* operations defined below */
          daddr_t   mt_count;       /* how many of them */
```

```
};

/* operations */
#define    MTWEOF    0              /* write an end-of-file record */
#detine    MTFSF     1              /* forward space file */
#define    MTFSR     3              /* forward space record */
#define    MTREW     5              /* rewind */
#define    MTNOP     7              /* no operation, sets status only */

/* structure for MTIOCGET - mag tape get status command */

struct     mtget     {
           short     mt_type;       /* type of magtape device */
/* the following six registers are very device dependent */
           short     mt_hard_error0; /* hard error byte 0 of status from DSD */
           short     mt_hard_error1; /* hard error byte 1 of status from DSD */
           short     mt_soft_error0; /* soft error byte of status from DSD */
           short     mt_at_bot;     /* byte 0xff when tape at bot */
           short     mt_retries;    /* byte number of retries by tape drive */
           short     mt_file_mark;  /* byte 0xff when file mark encountered */
/* end device-dependent registers */
           daddr_t mt_fileno;       /* file number of current position */
           daddr_t mt_blkno;        /* block number of current position */
};

/*
 * Constants for mt_type byte
 */
#define    MT_ISTS   0x01           /* Streaming Quarter Inch Tape Drive */

/* mag tape io control commands */
#define    MTIOCTOP  (('m'<<8)|1)   /* do a mag tape op */
#define    MTIOCGET  (('m'<<8)|2)   /* get tape status */

#ifndef KERNEL
#define    DEFTAPE   "/dev/rmtioctl" /* IOCTL device */
#endif
```

Each **read** or **write** call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. Each tape write will write one file mark on close and will either rewind or position itself at the file mark. Addition writes will be positioned after the file mark and can be accessed by using the *smt*(1) streaming tape manipulation program.

FILES

/dev/mt1 Cartridge tape
/dev/rmt1 Cartridge tape
/dev/rqic Cartridge tape
/dev/nrmt1 Cartridge tape — no rewind

The minor device numbers for each of the above devices to build special files using *mknod*(1) is based on the standard default minor device number being 0x00. The minor device number for the */dev/nrmt** and */dev/nrqic* is 0x01. The above minor device numbers refer to tape drive 0 of the DSD controller and at present the hardware only supports this one tape drive.

SEE ALSO

smt(1), tar(1), cpio(1)

BUGS

The status should be returned in a device independent format, but the status returned is very device independent.

NAME

stab – symbol table types

SYNOPSIS

#include <stab.h>

DESCRIPTION

Stab.h defines some values of the n_type field of the symbol table of a.out files. These are the types for permanent symbols (i.e. not local labels, etc.) used by the debugger *dbx*. Symbol table entries can be produced by the *.stabs* assembler directive. This allows one to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsigned long (usually an address). To avoid having to produce an explicit label for the address field, the *.stabd* directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the *.stabn* directive. The loader promises to preserve the order of symbol table entries produced by *.stab* directives. As described in *a.out*(4), an element of the symbol table consists of the following structure:

```
/*
 * Format of a symbol table entry.
 */
struct nlist {
        union {
                char   *n_name;     /* for use when in-core */
                long   n_strx;      /* index into file string table */
        } n_un;
        unsigned char  n_type;      /* type flag */
        char           n_other;     /* unused */
        short          n_desc;      /* see struct desc, below */
        unsigned n_value;           /* address or offset or line */
};
```

The low bits of the n_type field are used to place a symbol into at most one segment, according to the following masks, defined in *<a.out.h>*. A symbol can be in none of these segments by having none of these segment bits set.

```
/*
 * Simple values for n_type.
 */
#define N_UNDF  0x0   /* undefined */
#define N_ABS   0x2   /* absolute */
#define N_TEXT  0x4   /* text */
#define N_DATA  0x6   /* data */
#define N_BSS   0x8   /* bss */
```

```
#define N_EXT    01    /* external bit, or'ed in */
```

The n_value field of a symbol is relocated by the linker, *ld*(1) as an address within the appropriate segment. N_value fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the n_type field has one of the following bits set:

```
/*
 * Other permanent symbol table entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */

#define N_STAB        0xe0/* if any of these bits set, don't discard */
```

This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The symbolic debugger, dbx, uses the following n_type values:

```
#define N_GSYM    0x20    /* global symbol: name,,0,type,0 */
#define N_FNAME   0x22    /* procedure name (f77 kludge): name,,0 */
#define N_FUN     0x24    /* procedure: name,,0,linenumber,address */
#define N_STSYM   0x26    /* static symbol: name,,0,type,address */
#define N_LCSYM   0x28    /* .lcomm symbol: name,,0,type,address */
#define N_RSYM    0x40    /* register sym: name,,0,type,register */
#define N_SLINE   0x44    /* src line: 0,,0,linenumber,address */
#define N_SSYM    0x60    /* structure elt: name,,0, type,struct_offset */
#define N_SO      0x64    /* source file name: name,,0,0,address */
#define N_LSYM    0x80    /* local sym: name,,0,type,offset */
#define N_SOL     0x84    /* #included file name: name,,0,0,address */
#define N_PSYM    0xa0    /* parameter: name,,0,type,offset */
#define N_ENTRY   0xa4    /* alternate entry: name,linenumber,address */
#define N_LBRAC   0xc0    /* left bracket: 0,,0,nesting level,address */
#define N_RBRAC   0xe0    /* right bracket: 0,,0,nesting level,address */
#define N_BCOMM   0xe2    /* begin common: name,, */
#define N_ECOMM   0xe4    /* end common: name,, */
#define N_ECOML   0xe8    /* end common (local name): ,,address */
#define N_LENG    0xfe    /* second stab entry with length information */
```

where the comments give *dbx* conventional use for *.stabs* and the n_name, n_other, n_desc, and n_value fields of the given n_type. *Dbx* uses the n_desc field to hold a type specifier in the form used by the Portable C Compiler, *cc*(1), in which a base type is qualified in the following structure:

```
struct desc {
        short  q6:2,
               q5:2,
               q4:2,
```

```
                q3:2,
                q2:2,
                q1:2,
                basic:4;
};
```

There are four qualifications, with q1 the most significant and q6 the least significant:

0	none
1	pointer
2	function
3	array

The sixteen basic types are assigned as follows:

0	undefined
1	function argument
2	character
3	short
4	int
5	long
6	float
7	double
8	structure
9	union
10	enumeration
11	member of enumeration
12	unsigned character
13	unsigned short
14	unsigned int
15	unsigned long

SEE ALSO

as(1), ld(1), dbx(1), a.out(4)

NAME

ibtab – format of ibtab file

SYNOPSIS

/etc/ibtab

DESCRIPTION

The ibtab file is an unordered collection of entries, each of which describes one IEEE 488 bus node. Its format is understood by the *ibtab*(3) subroutine package, which is used by programs such as *iib*(1). The format and the package are loosely patterned after *fstab*(3,5), *getpio*(3), and *passwd*(5). Each entry consists of a line with colon-separated fields. Numeric items are interpreted according to the usual C rules: leading 0x implies hex, leading 0 implies octal, default decimal

FIELDS

File
: the ib file used to access the node.

Cfile
: the control file used to ioctl (control) the node.

Node
: the node (physical slot) number with respect to the control file. The host's node number is 0.

Flags
: flags pertaining to the node, as defined in the include file <*sys/ib_ioctl.h*>. This field may be numeric or symbolic, with symbols separated by the '|' character or white space. Possible flags are: SWAB, if the node does byte-reversed IEEE 488 bus i/o; VALID, if the node is for real; SRQ, if the node can assert SRQ; PPE, if the node can respond to parallel polls; PPC, if the node can be remotely configured for parallel polls; SC, if the node is system controller.

Tag
: the IEEE 488 bus address of the node.

Ppr
: the parallel poll response. The low-order 3 bits specify which line the node uses for parallel poll responses. This field is significant only if the IBN_PPC or IBN_PPE flags are present.

Comment
: ignored.

Lines beginning with the character are ignored.

EXAMPLE

```
# sample entry
# node 1, /dev/ib01
```

```
# controlled by /dev/ib00
# byte-reversed, can srq and ppoll
# gpib address 19
# ppoll line 3
/dev/ib01:/dev/ib00:1:VALID|SWAB|SRQ:19:03:board 0 node 1
```

FILES

SEE ALSO

ib(4), iib(1), ibtab(3)

NAME

autoconf – diagnostics from the autoconfiguration code

DESCRIPTION

When UNIX bootstraps it probes the innards of the machine it is running on and locates controllers, drives, and other devices, printing out what it finds on the console. This procedure is driven by a system configuration table which is processed by *config*(8) and compiled into each kernel.

MULTIBUS devices are located by probing to see if their control-status registers respond. If not, then the autoconfigure code will print out a message of the form "*xx* not installed". If the control status register responds but the device cannot be correctly initialized, a diagnostic warning will be printed on the console and the device will not be available to the system.

The variables *rootdev* and *swapdev* are used as prototypes in the kernel to specify where the system will find the root and swap devices, respectively. As each disk drive is attached to its controller, the systems checks for a match against the *rootdev* and *swapdev* variables.

If the given drive has a root partition on it (specified in the boot label) and the same unit number as stored in *rootdev* then it is chosen as the potential root drive. The **last** such drive found is used as the *rootdev* (thus ordering of the *config* file is important).

If the given drive has a swap partition on it (specified in the boot label) and the same unit number as stored in *swapdev* then it is chosen as the potential swap drive. The **last** such drive found is used as the *swapdev* (thus ordering of the *config* file is important).

SEE ALSO

intro(7), dklabel(7), config(8)

DIAGNOSTICS

%s%d at mbio 0x%04x ipl %d. This message is printed when probing a simple device or a controller. It means that the device successfully initialized itself (or lied about it anyway); is running at the printed multibus I/O address ("mbio") and will interrupt at priority level "ipl".

%s%d slave %d. The slave given drive (tape or disk) is attached as the printed slave to the *previous* controller printed.

%s%d not installed. The given device was not found on the multibus.

%s%d dead. The given device responded to its multibus address, but did not behave correctly. This might mean something is broken, or that two boards are wired with the same multibus i/o address. Typically, however, if two boards are mis address they won't probe at all, or the machine will hang trying to access them.

stray interrupt level %d. An interrupt occurred for which there is no interrupt service routine. A related message, **panic: default_intr**, *will follow this message*.

root on %s%d%c. Once all the controllers and devices have been probed, the autoconfigure routines will print out the chosen root device.

swap on %s%d%c [%dK]. Once all the controllers and devices have been probed, the autoconfigure routines will print out the chosen swap device as well as it size.

NAME

drum – paging device

DESCRIPTION

This file refers to the paging device in use by the system. This may actually be a subdevice of one of the disk drivers, but in a system with paging interleaved across multiple disk drives it provides an indirect driver for the multiple drives.

FILES

/dev/drum

BUGS

Reads from the drum are not allowed across the interleaving boundaries Since these only occur every .5Mbytes or so, and since the system never allocates blocks across the boundary, this is usually not a problem.

NAME

dsd – Qualogy 5217 st-506 disk/tape/floppy controller

SYNOPSIS

controller dsd0 at mb0 csr 0x7F00 priority 1 vector dsdintr
disk md0 at dsd0 drive 1 flags 0x00
disk md1 at dsd0 drive 2 flags 0x00

DESCRIPTION

This is a MULTIBUS st-506 disk and tape controller. The driver software supports two Winchester hard disk drives, one qic-02 streaming tape drive, and one floppy drive. This man page documents the hard disk support. See *qic*(7) for information on the streaming tape support and *floppy*(7) for the floppy support.

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with "md" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise *seek* calls should specify a multiple of 512 bytes.

The size of the various partitions supported by the driver in fact are a function of the drive itself. Present on each drive is a boot label which contains the partitions sizes and locations.

FILES

/dev/md[01][a-h] block files
/dev/rmd[01][a-h] raw files

SEE ALSO

autoconf(7), dklabel(7), qic(7), floppy(7)

DIAGNOSTICS

panic: dsdattach: geteblk0 failed. Can't happen. See a guru.
md%d (*No label***).** The named drive has no boot label and thus cannot be used.
(%s Name: %s). On a successful attach, the drive type is printed out followed by its "name" (a user specifiable name).
md%d%c: %s err(%s) at %d/%d/%d retry:%d

%d/%d/%d req%d atc%d. A hard error of some sort occurred.
dsdcmd: dsdbusy was not set to Zero(0)
panic: dsd: no status posted
dsd: cmd(%s), dev(%d), controller is hung on the bus
panic: dsd: couldn't start!
dsd: interrupt with empty queue
dsd: zero status
panic: dsdstatus
dsdstatus: dsdcmd failure
dsd: soft error: dev(%x). Any of the above errors indicates that something is seriously wrong with the controller.

BUGS

There are far too many printouts in the driver. There should be specific messages for specific problems.

NAME

duart – on board serial ports

SYNOPSIS

this device is automatically included; no config info is needed

DESCRIPTION

The on board serial ports for the IRIS system provide four serial ports, one of which is consumed supporting the IRIS keyboard. The remaining three ports support fully the rs 232 standard.

Each serial line attached to the back panel behaves as described in *termio*(7). Input and output for each line may independently be set to run at any of 16 speeds; see *termio*(7) for the encoding.

Unfortunately, due to a historical botch, the UNIX device names for the on board ports do not match the back panel definition. Unix uses a zero based numbering scheme, with port 0 being the keyboard. The back panel uses a one based numbering scheme.

FILES

/dev/ttyd[0-3]

SEE ALSO

termio(7)

NAME

floppy – Qualogy 5217 st-506 disk/tape/floppy controller

SYNOPSIS

controller dsd0 at mb0 csr 0x7F00 priority 1 vector dsdintr
disk md0 at dsd0 drive 3 flags 0x01

DESCRIPTION

This is a MULTIBUS st-506 disk and tape controller. The driver software supports 2 Winchester hard disk drives, one qic-02 tape drive, and one floppy. This manual page documents the floppy support. See *dsd*(7) for information on the Winchester disk support and *qic*(7) for the streaming tape support.

The block files access the disk via the system's normal buttering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise *seek* calls should specify a multiple of 512 bytes.

The minor device bits for the floppy device support several different sizes and configurations: single sided versus double sided; single density versus double density; 256 byte sectors versus 512 byte sectors. Each configuration can be combined, thus it is possible to use 256 byte single sided double density floppys (it may not be tested though).

FILES

/dev/floppy	block device
/dev/rfloppy	raw device

SEE ALSO

autoconf(7), dklabel(7), qic(7), dsd(7)

DIAGNOSTICS

mf%d: Write Protected. The floppy in the floppy drive is write protected.
mf%d: Is Diskette Formatted?. The controller is seriously confused by the floppy in the drive. Try using a formatted floppy.
%s on mf%d, slice %d. Usually proceeded by "out of space" when unix runs out of space on the drive.
mf%d%c: %s err(%s) at %d/%d/%d retry:%d %d/%d/%d req%d atc%d. A hard error of some sort occurred.
mf%d%c: cmd(%s), err(%s), physical block %d %d/%d/%d req%d atc%d. A hard error of some sort occurred.

NAME

iph – Interphase 2190 smd disk controller

SYNOPSIS

controller iph0 at mb0 csr 0x7010 priority 5 vector ipintr
disk ip0 at iph0 drive 0
disk ip1 at iph0 drive 1

DESCRIPTION

This is a generic MULTIBUS smd disk controller. Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with "ip" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise *seek* calls should specify a multiple of 512 bytes.

The size of the various partitions supported by the driver in fact are a function of the drive itself. Present on each drive is a boot label which contains the partitions sizes and locations.

FILES

/dev/ip[0-3][a-h] block files
/dev/rip[0-3][a-h] raw files

SEE ALSO

autoconf(7), dklabel(7)

DIAGNOSTICS

ip%d (*No label***).** The named drive has no boot label and thus cannot be used.
(%s Name: %s). On a successful attach, the drive type is printed out followed by its "name" (a user specifiable name). The name can be defined in the standalone utility **ipfex**.
ipintr: iptab.b_active == 0. A spurious interrupt from the controller occurred.
ipintr hard error(%x): %s block: %d cmd: %s. A hard error occurred while reading block %d.
ipcmd: timeout wait for status %x. While attempting to get status from the controller, a timeout occurred. The controller is probably hung.

ipcmd: status: %**x error:** %**x.** A hard error occurred during a non-interruptable command.
ipcmd: timeout waiting for cmd %**s to complete.** A command given to the drive in a non-interrupt fashion timed out.
%**s on ip**%**d, slice** %**d.** Usually printed by unix prefixed with the message "out of space".

NAME

mem, kmem – main memory

DESCRIPTION

Mem is a special file that is an image of the main memory of the computer. It may be used, for example, to examine (and even to patch) the system.

Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

To arrange the map in adb for examining kernel data structures use the command **?m e00400 effffff**.

FILES

/dev/mem
/dev/kmem

NAME

mem, kmem – main memory

DESCRIPTION

Mem is a special file that is an image of the main memory of the computer. It may be used, for example, to examine (and even to patch) the system.

Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

To arrange the map in adb for examining kernel data structures use the command **?m e00400 effffff**.

FILES

/dev/mem
/dev/kmem

NAME

null – data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

pty – pseudo terminal driver

SYNOPSIS

pseudo-device pty

DESCRIPTION

The *pty* driver provides support for a device-pair termed a *pseudo terminal*. A pseudo terminal is a pair of character devices, a *master* device and a *slave* device. The slave device provides processes an interface identical to that described in *termio*(7). However, whereas all other devices which provide the interface described in *termio*(7) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

FILES

/dev/pty[0-3]	master pseudo terminals
/dev/ttyp[0-3]	slave pseudo terminals

NAME

qic – Qualogy 5217 st-506 disk/tape controller

SYNOPSIS

controller dsd0 at mb0 csr 0x7F00 priority 1 vector dsdintr
tape md0 at dsd0 drive 0 flags 0x02

DESCRIPTION

This is a MULTIBUS ST-506 disk and tape controller. The driver software supports 2 Winchester hard disk drives, one qic-02 tape drive, and one floppy. This documents the tape support.

There is a 'raw' interface which provides for direct transmission between the tape and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.' Also, the **mtio(7)** interface to the drive is provided.

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise *seek* calls should specify a multiple of 512 bytes.

FILES

/dev/nrmt1	non rewinding drive
/dev/mt1	rewinding drive
/dev/rqic	traditional name
/dev/nrtape	non rewinding drive (yet another name)

SEE ALSO

autoconf(7), dklabel(7), qic(7), floppy(7)

DIAGNOSTICS

qic%d: No cartridge in drive. Can't write on nothing.
qic%d: Unit not ready. Something strange is going on.
qic%d: write protected. Tape must be writable if the drive is opened for read/write or write.
qic%d: couldn't rewind on open. up the driver.
panic: qicstart: couldn't start!. Controller is probably wedged.
qic%d: Hard error cmd %s No status available. Something is wrong with the tape drive such that the controller would hang if it attempted to read the drive status.
%s on qic%d. Were one to actually mount a filesystem on the tape drive, this is the message that would be printed if you ran out of space on the filesystem proceeded by "out of space".
qic%d: cannot read the Tape Status. Another way of saying that things are broken.
qic%d: cannot get Tape Status into memory. Yet another way of saying that things are really gosh darn awfully broken.
qic%d: Couldn't write file mark %d. For some reason, the drive was

unable to write a file mark.
Failed %d files. While skipping forward %d files, the tape drive returned an error. Usually means that the tape does not contain %d files.
Failed %d records. While skipping forward %d records, the tape drive returned an error. Usually means that the tape does not contain %d records.

BUGS

There are far too many printouts in the driver. More specific messages should be printed.

NAME

cib – ib driver control program

SYNOPSIS

cib [function [parameter ...]] ... < /dev/ib00

DESCRIPTION

Cib presents most of the special i/o control functions of the *ib*(4) driver in a useable (symbolic) form. The argument list is a series of function names and parameters, evaluated left-to-right. If the argument list is empty, *cib* reads and prints out the GPIB-796 registers.

The functions and their arguments are described below. Some are available only to the super-user (see *ib*(4)).

init
: executes the IBIOINIT ioctl.

setnode nodeno tag flags
: executes the IBIOSETNODE ioctl. Only the tag and flags fields are changed.

getnode nodeno
: executes the IBIOGETNODE ioctl and prints the structure in a readable form.

start
: executes the IBIOSTART ioctl.

ppc mask
: executes the IBIOPPC ioctl.

poll
: executes the IBIOPOLL ioctl.

srq value
: executes the IBIOSRQ ioctl.

takectl
: executes the IBIOTAKECTL ioctl.

passctl nodeno
: executes the IBIOPASSCTL ioctl.

cutoff
: executes the IBIOCUTOFF ioctl.

EXAMPLE

```
cib < /dev/ib00 \
        init \
        setnode 0 5 SWAB|PPE|SRQ\
        setnode 1 6 SWAB|PPE|SRQ\
        start
```

FILES

SEE ALSO

dib(8), ib(4), iib(1), tib(8).

NAME

dib – dump ib driver data structures

SYNOPSIS

dib [-c] [-t] [-n unix] [-u N] [-v]

DESCRIPTION

Dib is an aid for debugging the ib driver (see *ib*(4)). It prints out the current values of various kernel data structures used by the ib driver.

FLAGS

-c
: prints out per-minor-device input queues.

-f
: prints out buffer (freelist) information.

-n
: uses a different kernel namelist (default is */nunix*).

-u
: prints out structures relating to unit (board) N.

-v
: prints out driver globals.

The default is -c -f -n /nunix -u 0 -v . Good luck.

EXAMPLE

dib -u 1

FILES

/dev/kmem
/nunix

SEE ALSO

db(8), ib(4), iib(1), tib(8).

BUGS

Phase errors are possible.

NAME

tib – trace ib driver printouts

SYNOPSIS

tib [-l] [-n unix]

DESCRIPTION

Tib is an aid for debugging the ib driver (see *ib*(4)). It prints out trace information from the ib driver, essentially a history of print statements. Only the 300 most recent print statements are available. The -l flag shows each printed item on a separate line. The -n flag uses a different kernel namelist (default is */nunix*).

Printouts and/or tracing are enabled/disabled by patchable kernel variables. _ib_debug, _ib_dbg_debug, and _ib_tlc_debug, should be patched to non-0 values if debugging is desired. _ib_print_debug should be patched to non-0 for debugging printouts to the console. _ib_trc_debug should be patched to non-0 for debugging printouts to the trace buffer used by *tib*. All should be patched to 0 for maximum throughput.

Good luck.

EXAMPLE

```
#! /bin/csh
# script to turn off console printouts from ib driver
adb -w /nunix /dev/kmem << 'EOF'
_ib_print_debug/W 0
$q
'EOF'
```

FILES

/dev/kmem
/nunix

SEE ALSO

cib(8), dib(8), ib(4), iib(1).

BUGS

Phase errors are possible. The driver's throughput is reduced by ~40-50% when debugging is enabled.

Appendix D

Excelan Ethernet Board Address Change Procedure

D. Excelan Ethernet Board Address Change Procedure

D.1 Introduction

The Excelan ethernet board address must be changed prior to updating the IRIS 1400 workstation software to Release 2.1. The following instructions will assist you in completing this change. Before you begin the change, please read all of the instructions carefully. Use the check-off column to keep from getting out of step.

If any questions or problems should arise, please contact the **Geometry Hotline:**

SGI Geometry Hotline	
(800) 252-0222	North America except California (toll-free)
(800) 345-0222	California (toll-free)
(415) 962-0606	Worldwide (collect)

D.2 Requirements

The following is required:

Tools required:
One standard Phillips screwdriver One standard slotted screwdriver

Parts required:
Twelve (12) - Circuit board jumpers

Time required:
Thirty (30) minutes per system

D.3 Instructions

Check off	Step #	Instructions
	1	Reboot the IRIS 1400 Workstation. In super-user mode, type *reboot -q*.
	2	Power off the workstation.
	3	Remove the power cord from the rear of the workstation (Labeled *Power*). **WARNING:** This step is **required** for safety!
	4	Remove the front decor cover from the workstation. The cover is held in place by snap connectors. Use either of the following methods to remove the front cover: 1. Bang the heels of your hands against the sides of the cover at a 45° angle towards the front. 2. Tightly grip the top of the cover on both sides, and pull towards you. Place the cover aside.
	5	Remove the card cage cover from the circuit board card cage. The cover has a circuit card configuration sheet (which you may want to refer to during the course of the instructions) and is held into place by eight (8) Phillips-head screws. Loosen all of the screws, then slide the cover upward until the screw heads are positioned over the large holes, then pull the cover towards you. Put the cover aside.
	6	Remove the circuit board retainer bracket from the bottom of the card cage. Loosen the two (2) slotted screws from the bracket until free. Next, slide the bracket up approximately seven (7) inches until you reach the widest opening of card cage. Now, slide the bracket flush against the left wall of the card cage, then pull the right side of the bracket out of the enclosure. Put the bracket aside.

Check off	Step #	Instructions
	7	Remove the cable from the top connector (P3) of the PM2 circuit board located in slot one (1) of the card cage, (far left) **Note:** The other end of the cable is attached to top connector (P3) of the GF1 circuit board located in slot twenty (20) of the card cage. It need not be removed.
	8	Remove the RGB cable (small grey cable) from the middle connector (P2) of the DC3 circuit board located in slot eighteen (18) of the card cage.
	9	The Excelan board is static sensitive. Please take appropriate measures to reduce the possibility of electrostatic discharge (ESD) damage: 1. Discharge any static build up by using a grounded wrist strap while handling the board. 2. Handle the board by the edges only. 3. Place the board on a conductive surface while changing the address. Remove the Ethernet board located in slot two (2) of the card cage. As the board is being removed, remove the cable from the top connector (PI) on board. **Note:** Be sure all cables are clear of slot two (2) before removing the board.
	10	Place the Ethernet board on a cleared table with the components facing up and the card edge connector facing you. Refer to Figure 2 for address setting location. Insert the circuit board jumpers as referred to in Figure 1.
	11	Re-install the Ethernet board half way into slot two (2). Re-connect the cable to the Ethernet board and push the board the rest of the way into the slot. Be sure the board is firmly seated into its connector. If this is difficult, wedge the circuit board retainer bracket (from step 6) between a wall and the back of the unit and push hard with your thumbs on the clips at the top and bottom of the circuit board.
	12	Re-install the RGB cable to the middle connector (P2) of the PC3 circuit board located in slot eighteen (18).
	13	Re-install the cable to the top connector (P3) of the PM2 circuit board located in slot one (1).

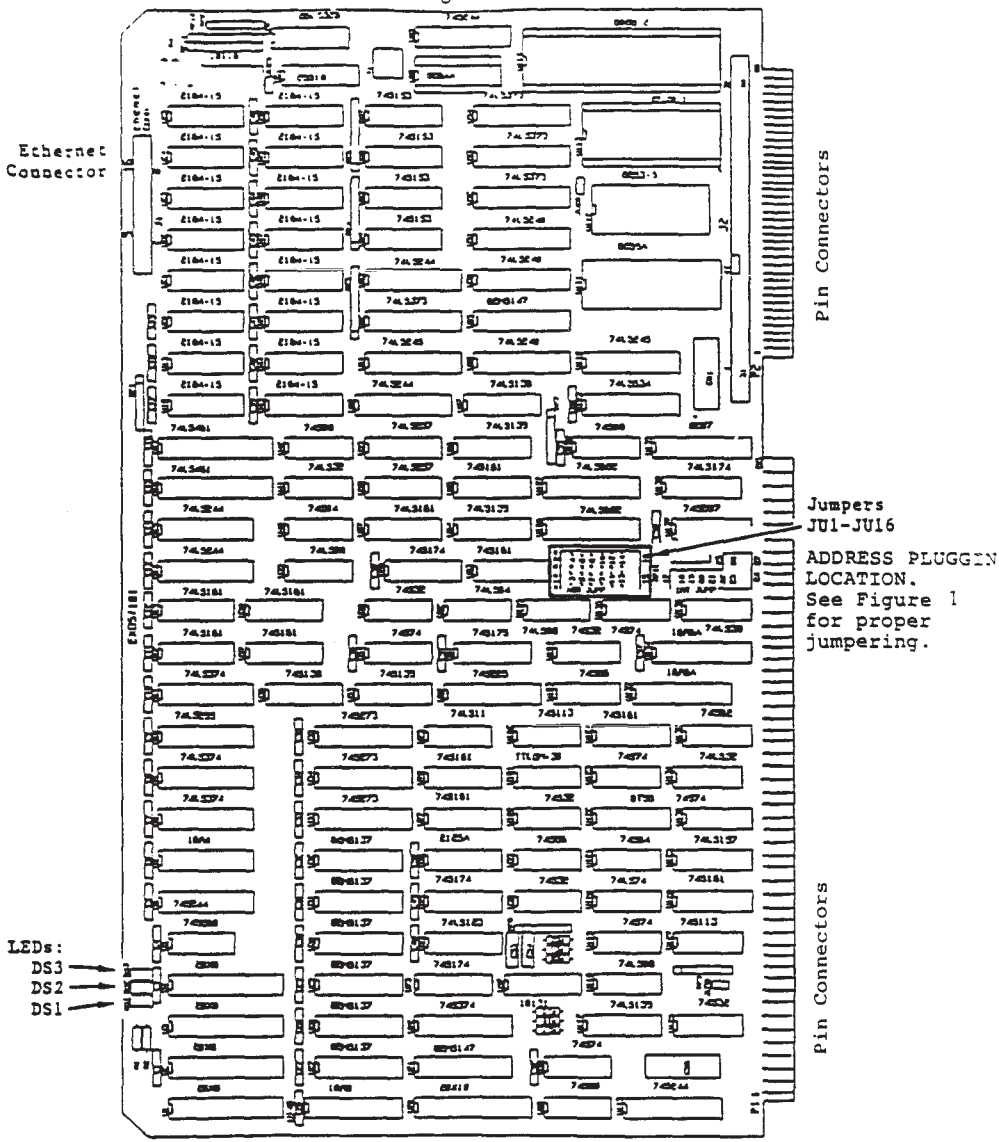
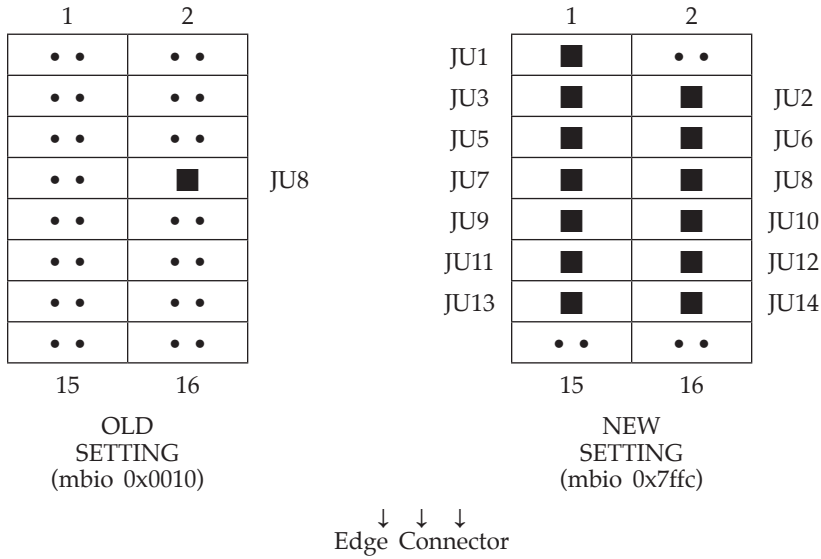

Ethernet Connector
Pin Connectors
Jumpers JU1-JU16
ADDRESS PLUGGIN LOCATION. See Figure 1 for proper jumpering.
LEDs:
DS3
DS2
DS1
Pin Connectors

Figure 2.

Check off	Step #	Instructions
	14	Re-install the circuit board retainer bracket.
	15	Re-install the card cage cover to the card cage.
	16	Re-install the decor cover of the workstation.
	17	Re-install the power cord at the rear of the workstation. (Labeled *Power*)
	18	Power on the workstation.
	19	Boot the UNIX system back up. Now that Release 2.1 software has been installed, the first reboot with Release 2.1 will indicate that the Ethernet board is installed, (*nx mbio 0x7ffc ipl2*)

D.4 Ethernet Address Jumpering



NOTE: The shaded areas of the drawings above are where the jumpers are inserted. See **Figure 2** for the location of the address pluggin location.

Figure 1.