ACADEMIC INFORMATION SYSTEMS 4.2 FOR THE IBM RT PC

ACADEMIC INFORMATION SYSTEMS 4.2

VOLUME II

Academic Information Systems 4.2 for the IBM RT PC

PRPQ #5799-CGZ, Release 2

Volume II

Academic Information Systems
International Business Machines Corporation
Palo Alto, CA

WARNING: Any shipment of Academic Information Systems 4.2 (hereinafter referred to as "4.2/RT") for the IBM RT PC to a country outside the United States requires a U.S. Government license.

Third Edition (December 1986)

Preface

This software and documentation is based in part on the 4.2 Berkeley Software Distribution under license from The Regents of the University of California. We gratefully acknowledge the following individuals and institutions for their role in its development: Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California; Individual Computing Systems, IBM Research, Yorktown Heights, New York; The IRIS Group, Brown University, Providence, Rhode Island; ITC, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

IBM
Academic Information Systems
Palo Alto, CA
March 1986

Preface to 4.2 Berkeley Software Distribution

This update to the 4.1 distribution of June 1981 provides support for the VAX[1] 11/730, full networking and interprocess communication support, an entirely new file system, and many other new features. It is certainly the most ambitious release of software ever prepared here and represents many man-years of work. Bill Shannon (both at DEC and at Sun Microsystems) and Robert Elz of the University of Melbourne contributed greatly to this distribution through new device drivers and painful debugging episodes. Rob Gurwitz of BBN wrote the initial version of the code upon which the current networking support is based. Eric Allman of Britton-Lee donated countless hours to the mail system. Bill Croft (both at SRI and Sun Microsystems) aided in the debugging and development of the networking facilities. Dennis Ritchie of Bell Laboratories also contributed greatly to this distribution, providing valuable advise and guidance. Helge Skriverik worked on the device drivers which enabled the distribution to be delivered with a TU58 console cassette and RX01 console floppy disk, and rewrote major portions of the standalone i/o system to support formatting of non-DEC peripherals.

Numerous others contributed their time and energy in organizing the user software for release, while many groups of people on campus suffered patiently through the low spots of development. As always, we are grateful to the UNIX[2] user community for encouragement and support.

Once again, the financial support of the Defense Advanced Research Projects Agency is gratefully acknowledged.

S. J. Leffler
W. N. Joy
M. K. McKusick

[1]VAX is a trademark of Digital Equipment Corporation.

[2]UNIX is a trademark of AT&T Bell Laboratories.

TRADEMARKS

CONTENTS

This page intentionally left blank.

VOLUME II. SUPPLEMENTARY DOCUMENTS

This volume provides information for configuring and operating 4.2/RT, as well as information for the programmer.

- Updates to three articles from the *UNIX Programmer's Manual,* Volume 2C:
  - **Operating Academic Information Systems 4.2**

    This article provides information helpful in using 4.2/RT.
  - **Building 4.2/RT Systems with Config**

    This article describes *config,* a tool used in building 4.2/RT system images, and provides information for using *config* on the IBM RT PC.
  - **Assembler Reference Manual for 4.2/RT**

    This article describes the usage and input syntax of *as*, the 4.2/RT assembler for the IBM RT PC.
- **Floating Point Arithmetic**

  This article summarizes floating point arithmetic in 4.2/RT.
- **The C Subroutine Interface for the IBM Academic Information Systems Experimental Display**

  This article describes a graphics interface for the experimental display.
- **Programmer's Notes**

  This article is a brief compendium of insights, suggestions, and notes gathered from the programmers who ported applications to 4.2/RT.
- **The IBM 3812 Pageprinter**

  This article provides information for installing the IBM 3812 Pageprinter, and installing and converting fonts.
- **4.2/RT Linkage Convention**

  This article describes the calling sequence used in 4.2/RT.
- **Recompiling with High C**

  This article provides guidance to C programmers who recompile existing programs with High C.
- **Professional Pascal Differences**

  This article describes significant differences between Professional Pascal and the Pascal compiler provided with 4.2BSD.
- **4.2/RT Console Emulators**

  This article explains the need for, and design of, emulators for 4.2/RT.
- **The Remote Virtual Disk System**

  This article describes a network service that provides a client computer with the appearance of removable-media disk drives and an unlimited number of disk packs.

Note that revised and new manual pages for Volume 1 of the *UNIX Programmer's Manual* are found in Volume I of this manual.

This page intentionally left blank.

Operating Academic Information Systems 4.2

ABSTRACT

This article is a revision of an article entitled "Installing and Operating 4.2BSD on the VAX," written in July 1983 by Samuel J. Leffler and William N. Joy, and found in Volume 2C of the *UNIX Programmer's Manual.* The revisions include additions and changes appropriate to the IBM RT PC. The following table summarizes organizational changes in the article.

Chapter in Original Article	Where Found in 4.2/RT Documentation
1. Introduction	Chapter 1 of this article
2. Bootstrap Procedures	Program Directory, Chapter 3, Installation Procedures
3. Upgrading from a 4BSD Release	Program Directory, Chapter 2, Saving Modified Files
4. System Setup	Chapter 2 of this article
5. Network Setup	Chapter 3 of this article
6. System Operation	Chapter 4 of this article
Appendix A. Bootstrap Details	Not applicable; dropped
Appendix B. Loading the Tape Monitor	Not applicable; dropped
Appendix C. Installation Troubleshooting	Not applicable; dropped
	Chapter 5. "AIX and 4.2/RT Co-residence" only in 4.2/RT

The article contains the following chapters:

1. **Introduction** provides information helpful in operating 4.2/RT. It covers hardware supported, distribution format, 4.2/RT device naming and 4.2/RT block and raw devices.

2. **System Setup** describes the procedures used to set up a 4.2/RT system.

3. **Network Setup** describes how to configure your system to use the networking support.

4. **System Operation** describes some typical 4.2/RT operations on an IBM RT PC.

5. **AIX and 4.2/RT Co-residence** describes installing and using AIX on a 4.2/RT system.

1. INTRODUCTION

This article provides information helpful in operating 4.2/RT.

1.1. Hardware Supported

4.2/RT runs on four models of the IBM RT PC: the IBM 6151 Model 010 and Model 015 (desk models) and the IBM 6150 Model 020 and Model 025 (floor models). This section describes the standard and optional features supported.

1.1.1. IBM 6151 Model 010 Processor

4.2/RT for the Model 010 desk model supports the following hardware:

- A 6151 System Unit and Keyboard with:
  - 2 Mb of memory
- An IBM RT Personal Computer 40 Mb Fixed-Disk Drive (#4735) (standard)
  - An IBM Personal Computer AT Fixed-Disk and Diskette (FD&D) Drive Adapter (#3428) (standard)
- An IBM Personal Computer AT High Capacity Diskette Drive (#0206) (standard)

1.1.2. IBM 6151 Model 015 Processor

4.2/RT for the Model 015 desk model supports the following hardware:

- A 6151 System Unit and Keyboard with:
  - 2 Mb of memory
- An IBM RT Personal Computer 70Mb Enhanced Small Device Interface (ESDI) Fixed-Disk Drive (#6941) (standard)
  - An ESDI Magnetic Media Adapter (#6341) (standard)
- An IBM Personal Computer AT High Capacity Diskette Drive (#0206) (standard)

1.1.3. IBM 6150 Model 020

4.2/RT for the Model 020 floor model supports the following hardware:

- A 6150 System Unit and Keyboard with:
  - 3 Mb of memory
  - 2 asynchronous (RS-232C) serial ports in the base unit (standard)
- An IBM RT Personal Computer 40 Mb Fixed-Disk Drive (#4735) (standard)
  - An IBM Personal Computer AT FD&D Drive Adapter (#3428) (standard)
- Up to two additional IBM RT Personal Computer Fixed-Disk Drives:
  - 1 or 2 with 40 Mb storage capacity (#4735) (requires an additional IBM Personal Computer AT FD&D Adapter (#3428) for the third drive)
  - 1 or 2 with 70 Mb storage capacity (#6941) (requires an additional ESDI Magnetic Media Adapter (#6341))

- An IBM Personal Computer AT High Capacity Diskette Drive (#0206) (standard)

1.1.4. IBM 6150 Model 025

4.2/RT for the Model 025 floor model supports the following hardware:

- A 6150 System Unit and Keyboard with:
    - 3 Mb of memory
    - 2 asynchronous (RS-232C) serial ports in the base unit (standard)
- An IBM RT Personal Computer 70 Mb ESDI Fixed-Disk Drive (#6941) (standard)
    - An ESDI Magnetic Media Adapter (#6341) (standard)
- Up to two additional IBM RT Personal Computer Fixed-Disk Drives:
    - 1 or 2 with 40 Mb storage capacity (#4735) (requires an additional IBM Personal Computer AT FD&D Adapter (#3428))
    - 1 or 2 with 70 Mb storage capacity (#6941) (requires an additional ESDI Magnetic Media Adapter (#6341) for the third drive)
- An IBM Personal Computer AT High Capacity Diskette Drive (#0206) (standard)

1.1.5. Peripherals and Optional Features

4.2/RT also supports the following on all four models:

- An IBM RT Personal Computer 1 Mb Memory Expansion (#8222) or an IBM RT Personal Computer 2 Mb Memory Expansion (#4739) or an IBM RT Personal Computer 4Mb Memory Expansion (#3156) (up to 8Mb total)
- Local area networks (up to two in each system unit):
    - IBM RT Personal Computer Token-Ring Network Adapter (#3797)
    - IBM RT Personal Computer Baseband Adapter (#6810) for use with Ethernet
- An IBM 6157 Streaming Tape Drive with IBM RT Personal Computer Streaming Tape Drive Adapter (#4797)
- IBM Academic Information Systems experimental display with adapter
- IBM 6153 Advanced Monochrome Graphics Display with IBM RT Personal Computer Advanced Monochrome Graphics Display Adapter (#4765)
- IBM 6154 Advanced Color Graphics Display with IBM RT Personal Computer Advanced Color Graphics Display Adapter (#4766)
- IBM 6155 Extended Monochrome Graphics Display with IBM RT Personal Computer Extended Monochrome Graphics Display Adapter (#4768)
- An IBM RT Personal Computer Four-Port Asynchronous RS-232C Adapter (#4763)
- An IBM 3812 Pageprinter
- An IBM RT Personal Computer Floating Point Accelerator (#4758)
- An IBM RT Personal Computer Mouse (#8426)

- An IBM Monochrome Display and Printer Adapter (#4900)
- An IBM 5151 Monochrome Display
- An IBM 5152 Graphics Printer
- An IBM 4201 Proprinter with adapter

1.2. Distribution Format

The distribution includes a cover letter and program directory, installation diskettes, distribution streaming tapes, and copies of this book, *Academic Information Systems 4.2 for the IBM RT PC.*

The 4.2/RT distribution includes all 4.2BSD source files, whether ported or not. (User-contributed software is not provided. The source distribution is available from the University of California at Berkeley.) However, the distribution includes binary files only for those programs that have been ported and tested. For more information on the files contained on the diskettes and tapes, see the Program Directory that accompanies distribution.

1.3. 4.2/RT Device Naming

Devices in 4.2/RT are typically given two- or three-letter names. Volume I, Section 4, of this book contains a complete list of the devices for which drivers are provided in 4.2/RT.

The normal standalone system that bootstraps the full 4.2/RT system uses two device names:

fd(y,z) for the floppy disk
hd(y,z) for the hard disk

The value *y* specifies the device. The value *z* is a (hard) disk partition (in the range 0-7).

A 4.2/RT physical (hard) disk is divided into eight logical disk partitions, each of which may occupy any consecutive cylinder range on the physical device. The cylinders occupied by the eight default partitions for each drive type are specified in the disk description file /etc/disktab (see *disktab*(5)). Non-standard partition sizes may be specified on a per disk basis (see *minidisk*(8R)). Each partition may be used either to store a 4.2/RT file system or as a raw data area (such as a paging area). Convention dictates the use of the first three partitions:

- The first partition (partition 0 on drive 0) stores a root file system from which 4.2/RT can be bootstrapped. The name of this partition is hd(0,0) for standalone programs, and /dev/hd0a for programs run under the kernel.
- The second partition -- hd(0,1) or /dev/hd0b -- is a paging area.
- The third partition -- hd(0,2) or /dev/hd0c -- allows access to the entire physical device.

  This partition can be used when making a backup copy. Such a backup must be restored onto the same disk from which it was backed up. Be extremely careful when you use this partition. The first cylinder contains bad sector forwarding and configuration information. If you overwrite these sectors, you will erase the bad block information.

  It is a good idea to copy the first 34 blocks of each disk in case of accident. The copy should be placed on a separate disk or diskette so it can be recovered if needed. The standalone COPY command can be used to copy and restore this information.

The remaining five partitions (numbered 3 through 7) can be used for additional mounted file systems. Partition 6 -- hd(0,6) or /dev/hd0g -- is normally used for the /usr mounted file system.

1.4. 4.2/RT Devices: Block and Raw

4.2/RT makes a distinction between *block* and *raw* (character) devices. Each disk has a block device interface that makes the device byte-addressable; you can write a single byte anywhere on the disk. The system reads the data from the disk sector, inserts the byte to be written, and writes the modified data. Names like /dev/hd0a indicate block devices. There are also raw devices available. These have names like /dev/rhd0a, the "r" here standing for "raw." The bootstrap procedure often uses the raw device interfaces, because these tend to work faster in some cases. In general, however, the block device interfaces are used.

Be aware that it is often important which interface is used: the character device interface (for efficiency), or the block device interface (to write specific bytes within a sector). Do not indiscriminately change the installation instructions to use the alternate type of device interface.

2. SYSTEM SETUP

This chapter describes procedures used to set up a 4.2/RT system. Use these procedures after you first install your system or when your system configuration changes.

2.1. Kernel Configuration

This section briefly describes the layout of the kernel code and how files for devices are made. For a full discussion of configuring and building system images, consult the article "Building 4.2/RT Systems with Config" later in this manual.

2.1.1. Kernel Organization

As distributed, the kernel source is in /usr/sys. The source may be physically located anywhere within any file system as long as a symbolic link to the location is created for the file /sys. (Many files in /usr/include are normally symbolic links relative to /sys.) In further discussions of the system source all path names will be given relative to /sys.

The directory /sys/sys contains the mainline machine-independent operating system code. Files within this directory are conventionally named with the following prefixes:

init_	system initialization
kern_	kernel (authentication, process management, etc.)
quota_	disk quotas
sys_	system calls and similar
tty_	terminal handling
ufs_	file system
uipc_	interprocess communication
vm_	virtual memory

The remaining directories are organized as follows.

/sys/h	machine-independent include files
/sys/conf	site configuration files and basic templates
/sys/net	network-independent, but network-related code
/sys/netinet	DARPA Internet code
/sys/netimp	IMP support code
/sys/netpup	PUP-1 support code
/sys/ca	IBM RT PC specific mainline code
/sys/caif	IBM RT PC network interface code
/sys/caio	IBM RT PC device drivers and related code
/sys/cacons	IBM RT PC console device drivers and related code

Many of these directories are referenced through /usr/include with symbolic links. For example, /usr/include/sys is a symbolic link to /sys/h. The system code as distributed is totally independent of the include files in /usr/include. This allows the system to be recompiled from scratch without the /usr file system mounted if the system sources have been relocated.

2.1.2. Devices and Device Drivers

Devices supported by 4.2/RT are implemented in the kernel by drivers whose source is kept in /sys/ca and /sys/caio. These drivers are loaded into the system when included in a cpu-specific configuration file kept in the conf directory. Devices are accessed through special files in the file system, made by the *mknod*(8) program, and normally kept in the /dev directory. For devices supported by the distribution system, files are created in /dev by the /dev/MAKEDEV shell script.

Determine the set of devices that you have and create a new /dev directory by running the MAKEDEV script. First create a new directory /newdev; copy MAKEDEV into it; edit the file MAKEDEV.local to provide an entry for local needs; run it to generate a /newdev directory. For instance, if your machine has one hard disk and a diskette, you would type:

```
#cd /
#mkdir newdev
#cp dev/MAKEDEV newdev/MAKEDEV
#cd newdev
#MAKEDEV hd0 fd0 std local
```

Note the "std" argument causes standard devices such as /dev/console (the machine console) to be created.

You can then type:

```
#cd /
#mv dev olddev ; mv newdev dev
#sync
```

to install the new device directory.

2.1.3. Building New System Images

The kernel configuration of each 4.2/RT system is described by a single configuration file, stored in the /sys/conf directory. To learn about the format of this file and the procedure used to build system images, you should:

- Read "Building 4.2/RT Systems with Config" later in this Volume.
- Study the manual pages for the devices you have (See Volume I of this manual or Volume 1 of the *UNIX Programmer's Manual*).
- Review the sample configuration file in the /sys/conf directory.

The configured system image "vmunix"[3] should be copied to the root and then booted to try it out. It is best to name it /newvmunix so as not to destroy the working system until you are sure it does work:

```
# cp vmunix /newvmunix
# sync
```

It is also a good idea to save the old system under some other name. In particular, we recommend that you save the generic distribution version of the system permanently as /genvmunix for use in emergencies.

To boot the new version of the system, power on the IBM RT PC. If it's already on, you can perform a hardware boot by using the *reboot*(8) command. Alternatively, you can use the *sync*(1) command, and then press and hold down the following keys:

<Ctrl> - <Alt> - <Pause>

2.2. Disk Configuration

This section describes how to lay out file systems to make use of the available space and to balance disk load for improved system performance.

[3]A system configured with the debugger is called "vmunix.ws".

2.2.1. Initializing /etc/fstab

Change into the directory /etc and copy the appropriate file from:

fstab.hd.1	(for a one-disk, desk model system)
fstab.hd.3	(for a three-disk, floor model system)

to the file /etc/fstab, i.e.:

```
# cd /etc
# cp fstab.hd.x fstab
```

where *x* is either 1 or 3.

This will set up the initial information about the usage of disk partitions.

2.2.2. Disk Naming and Divisions

Each physical disk drive can be divided into up to eight partitions; 4.2/RT typically uses only three or four partitions. The first partition (hd0a) stores a root file system from which 4.2/RT can be bootstrapped. The second partition is used for paging and swapping. The third partition allows access to the entire physical device.

The disk partition sizes for a drive are based on a set of four default partition tables. (See *diskpart*(8).) The particular table used depends on the size of the drive. The "a" partition is the same size across all drives, 15884 sectors. The "b" partition is 33440 sectors on 70-megabyte disks, and 10032 sectors on 40-megabyte disks. The "c" partition is large enough to access the entire disk, including the space at the front of the disk reserved for the bad sector forwarding table, and the space at the end of the disk containing the pool of replacement sectors.

Non-standard partition sizes may be specified on a per disk basis (see *minidisk*(8R)).

2.2.3. Space Available

The space available (in sectors) in the default disk partitions is listed in the following table.

Name	40 MB	70MB
hd?a	15884	15884
hd?b	10032	33440
hd?c	87040*	141372*
hd?d	15884	15884
hd?e	-----**	55936
hd?f	43826	19404
hd?g	59721	91476

* Note that a file system on the "c" partition can only be 138040 blocks on a 70-megabyte disk, or 86275 blocks on a 40-megabyte disk, to allow for the reserved space at the end of the disk.

** Partition "e" is not available on a 40-megabyte disk.

Be aware that the disks sizes are measured in disk sectors (512 bytes), while the 4.2/RT file system blocks are variably sized. User programs report disk space in kilobytes, and disk sizes in sectors. The /etc/disktab file used in making file systems specifies disk partition sizes in sectors. The default sector size of 512 bytes may be overridden with the "se" attribute.

2.2.4. Layout Considerations

There are several considerations in deciding how to arrange your disks. Two major considerations are adequate space and adequate throughput.

Many common system programs (C, the editor, the assembler, etc.) create intermediate files in the /tmp directory, so the file system where /tmp is stored should be large enough to accommodate most high-water marks. If you have several disks, mount /tmp in a root (i.e. first partition) file system on another disk. All programs that create files in /tmp also delete them but may leave dregs. Examine the directory periodically and delete old files.

On a single-disk system, there may not be sufficient free space on the root file system for /tmp. You can replace /tmp with a symbolic link to /usr/tmp on the hd0g partition, which should have sufficient space.

The efficiency with which 4.2/RT is able to use the CPU is often strongly affected by the configuration of disk controllers. For general time-sharing applications, the best strategy is to try to split the root file system (/), system binaries (/usr), the temporary files (/tmp), and the user files among several disk arms, and to interleave the paging activity among several arms.

It is critical for good performance to balance disk load. There are at least five components of disk load that you can divide between available disks:

1. The root (/) file system.
2. The /tmp file system.
3. The /usr file system.
4. The user files.
5. The paging activity.

The following possibilities are ones Berkeley has used when they have had two or three disks:

	disks	
what	2	3
/	1	2
tmp	1	3
usr	1	1
paging	1+2	1+3
users	2	2+3
archive	x	x

You should try to even out the disk load as much as possible by locating on separate arms those file systems between which heavy copying occurs. Note that long-term balancing of the load is not important; it is much more important to balance the load properly for when the system is busy.

Intelligent experimentation with a few file system arrangements can pay off in much improved performance. It is particularly easy to move the root, the /tmp file system and the paging areas. Place the user files and the /usr directory as space dictates, and experiment with the other, more easily-moved file systems.

2.2.5. File System Parameters

Each file system has associated parameters describing its block size, fragment size, and the geometry characteristics of the disk on which it resides. Inaccurate specification of the disk characteristics or haphazard choice of the file system parameters can cause substantial throughput degradation or significant wasted disk space. As distributed, file systems are configured according to the following table.

File system	Block size	Fragment size
/	8 Kbytes	1 Kbyte
usr	4 Kbytes	512 bytes
users	4 Kbytes	1 Kbyte

The root file system block size is made large to optimize bandwidth to the associated disk. This large block size is important because the /tmp directory is normally part of the root file. The large block size is also important because many of the most heavily used programs are demand-paged out of the /bin directory. The fragment size of 1 Kbyte is a "nominal" value to use with a file system. With a 1-Kbyte fragment size, disk ~ace utilization is approximately the same as with earlier versions of the file system.

The /usr file system uses a 4-Kbyte block size with 512-byte fragment size to achieve high performance while reducing the amount of space wasted by a larger fragment size. Space conservation is important here because the source code for the system is normally placed on this file system.

File systems for users have a 4-Kbyte block size with 1-Kbyte fragment size. These parameters have been selected based on the performance of Berkeley's user file systems. The 4-Kbyte block size provides adequate bandwidth while the 1-Kbyte fragment size provides acceptable space conservation and disk fragmentation.

You may chose other parameters in constructing file systems, but the factors involved in block size and fragment size are many and interact in complex ways. Larger block sizes result in better throughput to large files in the file system, because larger I/O requests can be performed. However, you should consider the average file sizes found in a file system and the performance of the internal system buffer cache. The system provides space in the inode for 12 direct block pointers, one single indirect block pointer, and one double indirect block pointer.[4] If a file uses only direct blocks, you can optimize access time to it by maximizing the block size. If a file spills over into an indirect block, increasing the block size of the file system may decrease the amount of space used (by eliminating the need to allocate an indirect block). However, if you increase the block size, and an indirect block is still required, the file will use more disk space (because indirect blocks are allocated according to the block size of the file system).

In selecting a fragment size for a file system, you must consider at least two things. The major performance tradeoffs are between an 8-Kbyte block file system and a 4-Kbyte block file system. Because of implementation constraints, the ratio of block size to fragment size cannot be greater than 8. An 8-Kbyte file system will always have a fragment size of at least 1 Kbyte. If a file system is created with a 4-Kbyte block size and a 1-Kbyte fragment size, and then upgraded to an 8-Kbyte block size and 1-Kbyte fragment size, identical space conservation occurs. However, if a file system has a 4-Kbyte block size and 512-byte fragment size, converting it to an 8K/1K file system causes significantly more space to be used. A 4-Kbyte block file system which might be upgraded to 8-Kbyte blocks for higher performance should use fragment sizes of at least 1 Kbyte to minimize the amount of work required in conversion.

A second, more important, consideration when selecting the fragment size for a file system is the level of fragmentation on the disk. With a 512-byte fragment size, storage fragmentation occurs much sooner, particularly with a busy file system running near full

[4] A triple indirect block pointer is also reserved, but not supported.

capacity. By comparison, the level of fragmentation in a 1-Kbyte fragment file system is much less severe. On file systems where many files are created and deleted, the 512-byte fragment size is more likely to result in apparent space exhaustion because of fragmentation. That is, when the file system is nearly full, file expansion that requires locating a contiguous area of disk space is more likely to fail on a 512-byte file system than on a 1-Kbyte file system. To minimize fragmentation problems of this sort, a parameter in the super block specifies a minimum acceptable free space threshold. When anyone but the super-user attempts to allocate disk space and the free space threshold is exceeded, the user is returned an error as if the file system were actually full. This parameter is nominally set to 10%, and can be changed by supplying a parameter to *newfs*, or by patching the super block of an existing file system.

In general, unless a file system is to be used for a special purpose application (for example, storing image processing data), Berkeley recommends using the default values supplied. Remember that the current implementation limits the block size to at most 8 Kbytes and the ratio of block size to fragment size must be in the range 1-8.

The disk geometry information used by the file system affects the block layout policies employed. The file /etc/disktab, as supplied, contains the data for drives supported by the system. When constructing a file system you should use the *newfs*(8) program and specify the type of disk on which the file system resides. This file also contains the default file system partition sizes, and default block and fragment sizes. To override any of the default values you can modify the file or use an option of *newfs*.

2.2.6. Implementing a Layout

To put a chosen disk layout into effect, use *newfs* (8) to create each new file system, and add its name to the file /etc/fstab. The system will check and mount each file system found in /etc/fstab when the system is bootstrapped.

Consider a system with 70-megabyte drives. On the first drive, (hd0), we put the root file system in hd0a and the /usr file system in hd0g. The /tmp directory was part of the root file system because no file system was mounted on /tmp. On a one-drive model, we put user files in the hd0g partition with the system binaries.

On a three-drive model, we created a file system in h1dg and put user files there, calling the file system /mnt. We interleaved the paging between the first and second drives. To do this we built a system configuration that specified:

```
config  vmunix root on hd0     swap on hd0 and hd1
```

to get the swap interleaved. We kept a backup copy of the root file system in the hd1a disk partition.

To make the /mnt file system we used the following commands:

```
#cd /dev
#MAKEDEV hd1
#newfs hd1g hd70r
(information about file system prints out)
#mkdir /mnt
#mount /dev/hd1g /mnt
```

2.3. Setup for Remote Virtual Disks (RVD)

For information regarding installing and setting up remote virtual disks, see the "The Remote Virtual Disk System" article.

2.4. Configuring Terminals

If 4.2/RT is to support simultaneous access from multiple terminals, you must edit the file /etc/ttys. (See *ttys*(5).)

Terminals connected to the system via RS232 ports are conventionally named **ttyxx** where xx identifies the specific line. The lines on 4-way RS232 cards are named /dev/tty00, /dev/tty01, . . . , /dev/tty15 (up to four cards may be installed). The planar serial ports are known as /dev/ttys0 and /dev/ttys1. The asynchronous communications cards are known as /dev/ttyc0 and /dev/ttyc1.

To add a new terminal, be sure the device is configured into the system and the special file for the device has been made by /dev/MAKEDEV. Then set the first character of the appropriate line of /etc/ttys to 1 (or add a new line).

The second character of each line in the /etc/ttys file lists the speed and initial parameter settings for the terminal. The commonly used choices are:

0	300-1200-150-110
2	9600
3	1200-300
5	300-1200

Here the first speed is the speed a terminal starts at; "break" switches speeds. Thus a newly added terminal /dev/tty00 could be added as

```
12tty00
```

if it were wired to run at 9600 baud. The definition of each terminal type is located in the file /etc/gettytab and read by the *getty*(8) program. To make custom terminal types, consult *gettytab*(5) before modifying this file.

Dialup terminals should be wired so that carrier is asserted only when the phone line is dialed up. For non-dialup terminals from which modem control is not available, you must either wire back the signals so that the carrier appears always to be present or show in the system configuration that the carrier is to be assumed to be present. See *asy*(4) and *psp*(4) for details.

You should also edit the file /etc/ttytype, placing the type of each new terminal there. (See *ttytype*(5).)

When the system is running in multi-user mode, all terminals are enabled that appear in /etc/ttys and have a 1 as the first character of their line. If, during normal operations, you want to disable a terminal line, you can edit the file /etc/ttys, changing the first character of the corresponding line to a 0. Then send a hangup signal to the *init*(8) process, using

```
# kill -1 1
```

Similarly, to enable a terminal, change the first character of a line from a 0 to a 1 and send a hangup signal to *init.*

Note that several programs, and /usr/src/etc/init.c in particular, will have to be recompiled if there are more than 100 terminals. Also note that if a special file is inaccessible when *init* tries to create a process for it, *init* will print a message on the console and try to reopen the terminal every minute, reprinting the warning message every ten minutes.

Finally note that you should change the names of any dialup terminals to ttyd? where ? is in the range [0-9a-f]; some programs use this property of the names to determine if a terminal is a dialup. You can put shell commands to do this in the /dev/MAKEDEV.local script.

While it is possible to use truly arbitrary strings for terminal names, the *ps* (1) command makes good use of the convention that tty names (by default, and also after dialups are named as suggested above) are distinct in the last 2 characters. We don't recommend you change this; the heuristic *ps* (1) uses that are based on these conventions may break down and *ps* will run MUCH slower.

2.5. Adding Users

You can add new users to the system by adding a line to the password file /etc/passwd. The procedure for adding a new user is described in *adduser* (8). You should add accounts for the initial user community, give each a directory and a password, and put users who wish to share software in the same group.

2.6. Site Tailoring

All programs that require the site name or some similar characteristic obtain the information through system calls or from files located in /etc. To supply a site name, edit the file /etc/rc.local. The first line in this file,

/bin/hostname *mysitename*

defines the value returned by the *gethostname* system call. Programs such as *getty*(8), *mail*(1), *wall*(1), and *who*(1) use this system call so that the binary images are site-independent.

2.7. Setting Up the Line Printer System

The line printer system consists of at least the following files and commands:

/usr/ucb/lpq	spooling queue examination program
/usr/ucb/lprm	program to delete jobs from a queue
/usr/ucb/lpr	program to enter a job in a printer queue
/etc/printcap	printer configuration and capability data base
/usr/lib/lpd	line printer daemon, scans spooling queues
/etc/lpc	line printer control program

The file /etc/printcap is a master data base describing both line printers directly attached to a machine and printers accessible across a network. The manual page *printcap*(5) describes the format of this data base and shows the default values for such things as the directory in which spooling is performed. The line printer system handles multiple printers, multiple spooling queues, local and remote printers, and printers attached via serial lines that require line initialization such as the baud rate.

Remote spooling via the network is handled with two spooling queues, one on the local machine and one on the remote machine. When a remote printer job is initiated with *lpr*, the job is queued locally and a daemon process is created to oversee the transfer of the job to the remote machine. If the destination machine is unreachable, the job will remain queued until it is possible to transfer the files to the spooling queue on the remote machine. The *lpq* program shows the contents of spool queues on both the local and remote machines.

To configure your line printers, consult the *printcap*(5) man page and the article entitled "4.2BSD Line Printer Spooler Manual" in Volume 2C of the *UNIX Programmer's Manual.* Include a call to *lpd*(8) in /etc/rc. (See also *ibm3812*(8) and *ppt*(8).)

2.8. Setting Up the Mail System

The mail system consists of the following commands:

/bin/mail	old standard mail program (from 32/V)
/usr/ucb/mail	UCB mail program, described in *mail*(1)
/usr/lib/sendmail	mail routing program
/usr/spool/mail	mail spooling directory
/usr/spool/secretmail	secure mail directory
/usr/bin/xsend	secure mail sender
/usr/bin/xget	secure mail receiver
/usr/lib/aliases	mail forwarding information
/usr/ucb/newaliases	command to rebuild binary forwarding database
/usr/ucb/biff	mail notification enabler
/etc/comsat	mail notification daemon
/etc/syslog	error message logger, used by sendmail

Normally, you use the *mail*(1) command to send and receive mail. This command provides a front end to edit the messages sent and received, and passes the messages to *sendmail*(8) for routing. To process each piece of mail, the routing algorithm uses knowledge of the network name syntax, aliasing and forwarding information, and network topology, as defined in the configuration file /usr/lib/sendmail.cf. The program /usr/bin/mail delivers local mail by adding it to the mailboxes in the directory /usr/spool/mail/*username*, using a locking protocol to avoid problems with simultaneous updates. After mail is delivered, the local mail delivery daemon /etc/comsat is notified, which in turn notifies users who have issued a "biff y" command that mail has arrived.

Normally, mail queued in the directory /usr/spool/mail can be read only by the recipient. To send mail that is secure against any possible perusal (except by a code-breaker), you should use the secret mail facility, which encrypts the mail so that no one can read it.

To set up the mail facility, read the instructions in the file READ_ME in the directory /usr/src/usr.lib/sendmail. Then adjust the necessary configuration files. You should also set up the file /usr/lib/aliases for your installation, creating mail groups as appropriate. Documents describing *sendmail*'s operation and installation appear in Volume 2C of the *UNIX Programmer's Manual.*

2.8.1. Setting Up a Uucp Connection

The version of *uucp* included in 4.2/RT is an enhanced version of that originally distributed with 32/V.[5] The enhancements include:

- support for many auto call units
- breakup of the spooling area into multiple subdirectories
- addition of an *L.cmds* file to control the set of commands that may be executed by a remote site
- enhanced "expect-send" sequence capabilities when logging in to a remote site
- new commands used to poll sites and obtain snapshots of *uucp* activity

This section gives a brief overview of *uucp* and points out the most important steps in its installation.

[5]The *uucp* included in this distribution is the result of work by many people; we gratefully acknowledge their contributions, but refrain from mentioning names in the interest of keeping this document current.

To connect two 4.2/RT machines with a *uucp* network link using modems, one site must have an automatic call unit and the other must have a dialup port. It is best if both sites have both.

You should first read the article "Uucp Implementation Description" in Volume 2B of the *UNIX Programmer's Manual.* It describes in detail the file formats and conventions, and will give you a little context. In addition, the document setup.tblms, located in the directory /usr/src/usr.bin/uucp/UUAIDS, may be of use in tailoring the software to your needs.

The *uucp* support is located in three major directories: /usr/bin, /usr/lib/uucp, and /usr/spool/uucp. User commands are kept in /usr/bin; operational commands are in /usr/lib/uucp; and /usr/spool/uucp is used as a spooling area. The commands in /usr/bin are:

/usr/bin/uucp	file copy command
/usr/bin/uux	remote execution command
/usr/bin/uusend	binary file transfer using mail
/usr/bin/uuencode	binary file encoder (for *uusend*)
/usr/bin/uudecode	binary file decoder (for *uusend*)
/usr/bin/uulog	scans session log files
/usr/bin/uusnap	gives a snapshot of *uucp* activity
/usr/bin/uupoll	polls remote system until an answer is received

The important files and commands in /usr/lib/uucp are:

/usr/lib/uucp/L-devices	list of dialers and hardwired lines
/usr/lib/uucp/L-dialcodes	dialcode abbreviations
/usr/lib/uucp/L.cmds	commands remote sites may execute
/usr/lib/uucp/L.sys	systems to communicate with, how to connect, and when
/usr/lib/uucp/SEQF	sequence numbering control file
/usr/lib/uucp/USERFILE	remote site pathname access specifications
/usr/lib/uucp/uuclean	cleans up garbage files in spool area
/usr/lib/uucp/uucico	*uucp* protocol daemon
/usr/lib/uucp/uuxqt	*uucp* remote execution server

while the spooling area contains the following important files and directories:

/usr/spool/uucp/C.	directory for command (C.) files
/usr/spool/uucp/D.	directory for data (D.) files
/usr/spool/uucp/X.	directory for command execution (X.) files
/usr/spool/uucp/D.*machine*	directory for local D. files
/usr/spool/uucp/D.*machine*X	directory for local X. files
/usr/spool/uucp/TM.	directory for temporary (TM.) files
/usr/spool/uucp/LOGFILE	log file of *uucp* activity
/usr/spool/uucp/SYSLOG	log file of *uucp* file transfers

To install *uucp* on your system, start by selecting a site name (less than eight characters). Next, create a *uucp* account in the /etc/passwd file and set up a password. Then, create the appropriate spooling directories with mode 755 and owned by user *uucp*, group *daemon.*

If you have an auto-call unit, create the L.sys, L-dialcodes, and L-devices files. The L.sys file should contain the phone numbers and login sequences required to establish a connection with a *uucp* daemon on another machine. For example, the Berkeley L.sys file looks something like:

```
adiron Any ACU 1200 out0123456789- ogin-EOT-ogin uucp
cbosg Never Slave 300
cbosgd Never Slave 300
chico Never Slave 1200 out2010123456
```

The first field is the name of a site; the second tells when the machine may be called; the third specifies how the host is connected (through an ACU, a hardwired line, etc.); the fourth is the phone number to use in connecting through an auto-call unit; and the fifth is a login sequence. The phone number may contain common abbreviations that are defined in the L-dialcodes file. The device specification should refer to devices found in the L-devices file. Using only ACU causes the *uucp* daemon, *uucico*, to search for any available auto-call unit in L-devices. Berkeley's L-dialcodes file is of the form:

```
ucb 2
out 9%
```

while their L-devices file is:

```
ACU cul0 unused 1200 ventel
```

Refer to the README file in the *uucp* source directory for more information about installation.

As *uucp* operates, it creates (and removes) many small files in the directories underneath /usr/spool/uucp. Sometimes files are left undeleted; purge them with the *uuclean* program. The log files can grow without bound unless trimmed back; use *uulog* to maintain these files. Many useful aids in maintaining your *uucp* installation are included in a subdirectory UUAIDS beneath /usr/src/usr.bin/uucp. Peruse this directory, and read the "setup" instructions also located there.

3. NETWORK SETUP

4.2/RT provides support for the DARPA standard Internet protocols IP, ICMP, TCP, and UDP. These protocols may be used on top of a variety of hardware devices ranging from the IMPs used in the ARPANET to local area network controllers for the Ethernet. Network services are split between the kernel (communication protocols) and user programs (user services such as TELNET and FTP). This section describes how to configure your system to use the networking support.

3.1. System Configuration

To configure the kernel to include the Internet communication protocols, define the INET option and include the pseudo-devices "inet", "pty", and "loop" in your machine's configuration file. The "pty" pseudo-device forces the pseudo terminal device driver to be configured into the system. See *pty*(4). The "loop" pseudo-device forces inclusion of the software loopback interface driver. The loop driver is used in network testing and also by the mail system.

If you are planning to use the network facilities on a 10Mb/s Ethernet or on the IBM Token-Ring local area network, the pseudo-device "ether" should also be included in the configuration; this forces inclusion of the Address Resolution Protocol module used in mapping between 48-bit Ethernet or 48-bit Token-Ring addresses and 32-bit Internet addresses.

Also, if you have an imp, you must include the pseudo-device "imp". Note that no hardware drivers are provided that support imp.

Before configuring the appropriate networking hardware, you should consult the manual pages in Volume I, Section 4, of this book. Software support exists for the device "un," the IBM Baseband Adapter for use with Ethernet network, and for the device "lan", the IBM Token-Ring local area network interface.

Network interface drivers require some or all of their host address to be defined at boot time. This is accomplished with *ifconfig*(8C) commands included in the /etc/rc.local file. Interfaces that can dynamically deduce the host part of an address but not the network number take the network number from the address specified with *ifconfig*. Hosts that use a more complex address mapping scheme, such as the Address Resolution Protocol, *arp*(4), require the full address. The manual page for each network interface describes the method used to establish a host's address. *Ifconfig*(8) can also set options for the interface at boot time. These options include disabling the use of the Address Resolution Protocol and/or the use of trailer encapsulation. These options are useful if a network is shared with hosts running software that is unable to perform these functions. Options are set independently for each interface and apply to all packets sent using that interface. An alternative approach to ARP is to divide the address range, using ARP only for those addresses below the cutoff and using another mapping above this constant address. See the source (/sys/netinet/if_ether.c) for more information.

To use the pseudo terminals just configured, device entries must be created in the /dev directory. To create 16 pseudo terminals (plenty, unless you have a heavy network load) execute the following commands:

```
# cd /dev
# MAKEDEV pty0
```

More pseudo terminals may be made by specifying *pty1*, *pty2*, etc. The kernel normally includes support for 32 pseudo terminals unless the configuration file specifies a different number. Each pseudo terminal actually consists of two files in /dev, a master and a slave. The master pseudo terminal file is named /dev/pty? and the slave is /dev/ttyp?. Pseudo terminals are also used by the *script*(1) program. In addition to creating the pseudo terminals, be sure to install them in the /etc/ttys file (with a '0' in the first column so no *getty* is started) and in the /etc/ttytype file (with type "network").

When configuring multiple networks, some thought must be given to the ordering of the devices in the configuration file. The first network interface configured in the system is used as the default network when the system is forced to assign a local address to a socket. This means that your most widely known network should always be placed first in the configuration file.

3.2. Network Data Bases

Several data files are used by the network library routines and server programs. Most of these files are host-independent and updated only rarely.

File	Manual reference	Use
/etc/hosts	*hosts*(5)	host names
/etc/networks	*networks*(5)	network names
/etc/services	*services*(5)	list of known services
/etc/protocols	*protocols*(5)	protocol names
/etc/hosts.equiv	*rshd*(8C)	list of "trusted" hosts
/etc/rc.local	*rc*(8)	command script for starting servers
/etc/ftpusers	*ftpd*(8C)	list of "unwelcome" ftp users

The files distributed are set up for the ARPANET or other Internet hosts. Local networks and hosts should be added to describe the local configuration. The Berkeley entries may serve as examples; also see the next section. You must choose network numbers for each Ethernet. For sites not connected to the Internet, these numbers can be chosen arbitrarily; otherwise, you should use the normal channels for allocating network numbers.

3.2.1. Regenerating /etc/hosts and /etc/networks

The host and network name data bases are normally derived from a file retrieved from the Internet Network Information Center at SRI. (Note that it is your responsibility to be able to connect to the Internet Network Information Center.) Use the program /etc/gettable to retrieve the NIC host data base and the program /etc/htable to convert it to the format used by the libraries.

```
# cd /usr/src/ucb/netser/htable
# /etc/gettable sri-nic
Connection to sri-nic opened.
Host table received.
Connection to sri-nic closed.
# /etc/htable hosts.txt
Warning, no localgateways file.
#
```

The *htable* program generates two files of interest in the local directory: hosts and networks. If a localhosts file is present in the working directory, its contents are first copied to the output file. Similarly, a localnetworks file may be prepended to the output created by *htable*. It is usually wise to run *diff*(1) on the new host and network data bases before installing them in /etc.

3.2.2. /etc/hosts.equiv

The remote login and shell servers use an authentication scheme based on trusted hosts. The hosts.equiv file contains a list of hosts that are considered trusted and/or under a single administrative control. When a user contacts a remote login or shell server requesting service, the client process passes the user's name and the official name of the host on which the client is located. In the simple case, if the host's name is located in hosts.equiv and the user has an account on the server's machine, then service is rendered (i.e. the user is allowed to log in, or the command is executed). Users may constrain this "equivalence" of machines by installing a .rhosts file in their login directories. The root login is handled specially, bypassing the hosts.equiv file and using only the /.rhosts file.

Thus, to create a class of equivalent machines, the hosts.equiv file should contain the *official* names for those machines. For example, most machines on Berkeley's major local network are considered trusted, so the hosts.equiv file is of the form:

```
ucbarpa
ucbcalder
ucbdali
ucbernie
ucbkim
ucbmatisse
ucbmonet
ucbvax
ucbmiro
ucbdegas
```

3.2.3. /etc/rc.local

Most network servers are automatically started up at boot time by the command file /etc/rc (if they are installed in their presumed locations). These include the following:

/etc/rshd	shell server
/etc/rexecd	exec server
/etc/rlogind	login server
/etc/rwhod	system status daemon

To have other network servers started up as well, commands of the following sort should be placed in the site-dependent file /etc/rc.local.

```
if [ -f /etc/telnetd ]; then
        /etc/telnetd & echo -n ' telnetd'                >/dev/console
fi
```

The following servers are included with the system and should be installed in /etc/rc.local as the need arises.

/etc/telnetd	TELNET server
/etc/ftpd	FTP server
/etc/tftpd	TFTP server
/etc/syslog	error logging server
/etc/sendmail	SMTP server
/etc/courierd	Courier remote procedure call server
/etc/routed	routing table management daemon
/etc/landump	IBM Token-Ring diagnostic daemon

Consult the manual pages and accompanying documentation (particularly for sendmail) for details about their operation.

3.2.4. /etc/ftpusers

The FTP server included in the system provides support for an anonymous FTP account. Because of the inherent security problems with such a facility you should read this section carefully if you consider providing such a service.

An anonymous account is enabled by creating a user *ftp*. When a client uses the anonymous account a *chroot*(2) system call is performed by the server to restrict the client from moving outside that part of the file system where the user ftp home directory is located. Because a chroot call is used, certain programs and files must be supplied to the server process for it to execute properly. Further, one must be sure that all directories and executable images are unwritable. The following directory setup is recommended:

```
# cd ~ftp
# chmod 555 .; chown ftp .; chgrp ftp .
# mkdir bin etc pub
# chown root bin etc
# chmod 555 bin etc
# chown ftp pub
# chmod 777 pub
# cd bin
# cp /bin/sh /bin/ls .
# chmod 111 sh ls
# cd ../etc
# cp /etc/passwd /etc/group .
# chmod 444 passwd group
```

When a local user wishes to place files in the anonymous area, they must be placed in a subdirectory. In the Berkeley setup, the directory ~ftp/pub is used.

Aside from the problems of directory modes, the ftp server may provide a loophole for interlopers if certain user accounts are allowed. The file /etc/ftpusers is checked on each connection. If the requested user name is located in the file, the request for service is denied. This file normally has the following names on Berkeley systems:

```
uucp
root
```

Accounts with nonstandard shells and no passwords (e.g., who or finger) should also be listed in this file to prevent their use as anonymous accounts with ftp.

3.3. Routing, Gateways, and Bridges

If your environment allows access to networks not directly attached to your host, you need to set up routing information to allow packets to be properly routed. Two schemes are supported by the system. The first scheme employs the routing table management daemon /etc/routed to maintain the system routing tables. The routing daemon uses a variant of the Xerox Routing Information Protocol to maintain up-to-date routing tables in a cluster of local area networks. By using the /etc/gateways file created by /etc/htable, the routing daemon can also initialize static routes to distant networks. When the routing daemon is started up (usually from /etc/rc.local), it reads /etc/gateways and installs those routes defined there, and then broadcasts on each local network to which the host is attached to find other instances of the routing daemon. If any responses are received, the routing daemons cooperate in maintaining a globally consistent view of routing in the local environment. This view can be extended to include remote sites also running the routing daemon by setting up suitable entries in /etc/gateways. Consult *routed*(8C) for a more thorough discussion.

The second approach is to define a wildcard route to a smart gateway and depend on the gateway to provide ICMP routing redirect information that will dynamically create a routing data base. This is done by adding an entry of the form

```
/etc/route add 0 smart-gateway 1
```

to /etc/rc.local. See *route*(8C) for more information. The wildcard route, shown by a 0-value destination, will be used by the system as a "last resort" in routing packets to their destination. Assuming the gateway to which packets are directed is able to generate the proper routing redirect messages, the system will then add routing table entries based on the information supplied. This approach has certain advantages over the routing daemon but is unsuitable in an environment where there are only bridges (i.e. pseudo gateways that, for instance, do not generate routing redirect messages). Further, if the smart gateway goes down, the only alternative for maintaining service is manually altering the routing table entry.

The system always listens to and processes routing table redirect information, so it is possible to combine both the above facilities. For example, the routing table management process might be used to maintain up-to-date information about routes to geographically local networks, while employing the wildcard routing techniques for "distant" networks. The *netstat*(1) program may be used to display routing table contents as well as various routing oriented statistics. For example,

```
#netstat −r
```

will display the contents of the routing tables, while

```
#netstat −r −s
```

will show the number of routing table entries dynamically created as a result of routing redirect messages, etc.

4. SYSTEM OPERATION

This section describes some typical 4.2/RT operations on an IBM RT PC, including:

- Bootstrapping and shutdown
- Checking system and device error logs
- Checking file systems and performing backups
- Moving file systems
- Monitoring system performance
- Recompiling and reinstalling system software
- Making local modifications
- Accounting for connect time and process resources
- Controlling resources
- Network troubleshooting
- Monitoring specific files

Procedures described here are used periodically to reboot the system, analyze error messages from devices, do disk backups, monitor system performance, recompile system software and control local changes.

4.1. Bootstrapping and Shutdown

During a normal reboot, the system checks the disks and comes up in multi-user mode without intervention at the console. To bring the system up in single-user mode, press and hold down **<Ctrl>-<C>** as soon as the system prints the date. This interrupts the boot with only the console terminal active.

To boot from the console, press and hold down the following keys:

<Ctrl>-<Alt>-<Pause>

The system tries to boot from a diskette, then from the hard disk.

You can also boot in single-user mode by explicitly typing the system name in response to the boot prompt:

```
:hd(0,0)vmunix
```

To bring the system up to a multi-user mode from single-user mode, press and hold down **<Ctrl>-<D>** on the console. The system executes /etc/rc (a multi-user restart script) and /etc/rc.local, and comes up on the terminals listed as active in the file /etc/ttys. See *init*(8) and *ttys*(5). Note, however, that this does not do a file system check. If the previous shutdown was not clean, you should run "fsck −p" or force a reboot with *reboot*(8) to check the disks.

When the system is in multi-user mode, you can take it to single-user mode with either:

```
# kill 1
```

or the *shutdown*(8) command. The latter is much more polite if there are other users logged in. When you are in multi-user mode, either command will kill all processes and give you a shell on the console, as if you had just booted. File systems remain mounted after the system becomes single-user. To change to multi-user mode again, use the following commands:

```
# cd /
# /etc/umount -a
# <Ctrl>-<D>
```

Note that the file /usr/adm/shutdownlog records each system shutdown, crash, processor halt, and reboot, with its associated cause.

4.2. Checking System and Device Error Logs

When errors occur on peripherals or in the system, the system displays a warning diagnostic on the console. These messages are collected regularly and written into the system error log file /usr/adm/messages.

Error messages issued by the devices in the system are described with the drivers for the devices in Volume I, Section 4, of this manual. If errors occur suggesting hardware problems, you should contact your hardware support group. You should check the error log file regularly, using the command:

```
tail −r /usr/adm/messages
```

4.3. Checking File Systems and Performing Backups

You should periodically check all file systems for consistency. Use the *fsck*(1) command weekly in the absence of problems, and always (usually automatically) after a crash. You can use the procedures of *reboot*(8) to put the system in a state where a file system check can be performed manually or automatically.

You should also back up file systems regularly. Use *dump*(8) for both complete and incremental dumps. Berkeley recommends a towers-of-hanoi dump sequence with full dumps taken every month.

Use three sets of dump media (streaming tape): daily, weekly, and monthly. Perform daily dumps circularly on the daily set with sequence '3 2 5 4 7 6 9 8 9 9 9 . . .' Each weekly is a level 1; daily dump sequence levels restart after each weekly dump. Full dumps are level 0; daily sequence levels also restart after each full dump.

Thus a typical dump sequence would be:

Dump ID	Level Number	Date	Opr	Size
FULL	0	Nov 24, 1984	sy	137K
D1	3	Nov 28, 1984	sy	29K
D2	2	Nov 29, 1984	ac	34K
D3	5	Nov 30, 1984	ac	19K
D4	4	Dec 1, 1984	ac	22K
W1	1	Dec 2, 1984	sy	40K
D5	3	Dec 4, 1984	ac	15K
D6	2	Dec 5, 1984	sy	25K
D7	5	Dec 6, 1984	sy	15K
D8	4	Dec 7, 1984	ac	19K
W2	1	Dec 9, 1984	sy	118K
D9	3	Dec 11, 1984	ac	15K
D10	2	Dec 12, 1984	sy	26K
D1	5	Dec 15, 1984	ac	14K
W3	1	Dec 17, 1984	sy	71K
D2	3	Dec 18, 1984	sy	13K
FULL	0	Dec 22, 1984	ac	135K

Take weekly dumps often enough that daily dumps always fit on one streaming tape, and never get to the sequence of 9's in the daily level numbers.

Operators can execute **/etc/dump w** at login to learn what needs to be dumped (based on the /etc/fstab information). Be sure to create a group **operator** in the file /etc/group so that *dump*(8) can notify logged-in operators when it needs help.

Dumping files by name is best done with *tar*(1) but the amount of data moved is limited to a single tape. If there are enough drives, you can copy entire disks with *dd*(1) using the raw special files and an appropriate blocking factor. The number of sectors per track is usually a good value to use; consult /etc/disktab.

You should also make full dumps of the root file system on a regular schedule. This is especially true on a system with only one disk. If the root file system is damaged by a hardware or software failure, you can rebuild a workable disk by restoring the dump.

Exhaustion of user file space is certain to occur now and then. You can impose disk quotas, or you might use the programs *du*(1), *df*(1), and *quot*(8) combined with messages of the day and personal letters.

4.4. Moving File Systems

If you have a streaming tape, the best way to move a file system is to dump it to tape using *dump*(8), create a new file system using *newfs*(8), and restore the tape using *restore*(8). If you do not have tape, *dump* accepts an argument telling where to put the dump; you might use another disk. The restore program uses an "in-place" algorithm that allows file system dumps to be restored without concern for the original size of the file system. Further, portions of a file system may be selectively restored in a manner similar to the tape archive program.

To merge a file system into an existing one, use *tar*(1).

To shrink a file system, dump the original and restore it onto the new file system. To shrink the root file system with only one disk drive, the procedure is more complicated:

(1) Dump the root file system to a remote streaming tape using *rdump*(8).

(2) Bring the system down.

(3) Load the distribution tape and install the new root file system as you did when first installing the system.

(4) Boot normally using the newly-created disk file system.

Note that if you change the disk partition tables or add new disk drivers, you must modify the default disk partition tables in /etc/disktab and add the drivers to the standalone system in /sys/standca.

4.5. Monitoring System Performance

The *vmstat* program provided with the system is designed to be an aid to monitoring system-wide activity. Together with the *ps*(1) command (as in "ps av"), it can be used to investigate system-wide virtual memory activity. By running *vmstat* when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, paging and swapping activity, and disk and cpu utilization. Ideally, there should be few blocked (b) jobs; little paging or swapping activity; available bandwidth on the disk devices (most single arms peak out at 30-35 tps in practice); and high (above 60%) user cpu utilization (us).

If the system is busy, then the count of active jobs may be large, and several of these jobs may often be blocked (b). If the virtual memory is active, then the paging daemon will be running (sr will be non-zero). It is healthy for the paging daemon to free pages when the virtual memory gets active; it is triggered by the amount of free memory dropping below a threshold and increases its pace as free memory goes to zero.

If you run *vmstat* when the system is busy (a "vmstat 1" gives all the numbers computed by the system), you can find imbalances by noting abnormal job distributions. If many processes are blocked (b), then the disk subsystem is overloaded or imbalanced. If you have several non-dma devices or open teletype lines that are "ringing," or user programs that are doing high-speed non-buffered input/output, then the system time may go high (60-70% or higher). It is often possible to pin down the cause of high system time by seeing if there is excessive context switching (cs), interrupt activity (in) or system call activity (sy).

If the system is heavily loaded or if you have little memory for your load (1 megabyte is little in most any case), then the system may be forced to swap. This is likely to be accompanied by a noticeable reduction in system performance and pauses when interactive jobs such as editors swap out.

4.6. Recompiling and Reinstalling System Software

It is easy to regenerate the system, and it is a good idea to try rebuilding pieces of the system to build confidence in the procedures. The system consists of two major parts: the kernel itself (/sys) and the user programs (/usr/src and subdirectories). The major part of this is /usr/src.

The major library is the C library in /usr/src/lib/libc. The library is remade by changing into the correct directory and typing:

```
# make
```

and then installed by typing:

```
# make install
```

Similarly, typing:

```
# make clean
```

cleans up.

NOTE: The code to support IEEE floating point emulation is distributed only in object form on the system. If the system is rebuilt, be very careful not to delete the */usr/src/lib/libc/ca/gen/xFP*.o* modules. It is strongly recommended that you **tar** these modules to a diskette before starting a rebuild.

The source for all other libraries is kept in subdirectories of /usr/src/usr.lib; each has a makefile and can be recompiled by the above recipe.

If you look at /usr/src/Makefile, you will see that you can recompile the entire system source with one command. To recompile a specific program, find out where the source resides with the *whereis*(1) command, then change to that directory and remake it with the makefile present in the directory. For instance, to recompile "date", all one has to type is:

```
# whereis date
date: /usr/src/bin/date.c /bin/date /usr/man/man1/date.1
# cd /usr/src/bin
# make date
```

This will create an unstripped version of the binary of "date" in the current directory. To install the binary image, use the install command:

```
# install -s date /bin/date
```

The −s option will insure the installed version of date has its symbol table stripped. The install command should be used instead of mv or cp as it understands how to install programs even when the program is currently in use.

If you wish to recompile and install all programs in a particular target area, you can override the default target by typing:

```
# make
# make DESTDIR =pathname install
```

To regenerate all the system source you can type:

```
# cd /usr/src
# make
```

If you modify the C library (perhaps to change a system call) and want to rebuild and install everything from scratch, you have to be careful. You must insure the libraries are installed before the remainder of the source; otherwise the loaded images will not contain the new routine from the library. The following steps are recommended:

```
# cd /usr/src
# cd lib; make install
# cd ..
# make usr.lib
# cd usr.lib; make install
# cd ..
# make bin etc usr.bin ucb games local ibm
# for i in bin etc usr.bin ucb games local ibm; do (cd $i; make install); done
```

4.7. Making Local Modifications

To keep track of changes to system source, Berkeley migrates changed versions of commands in /usr/src/bin, /usr/src/usr.bin, and /usr/src/ucb in through the directory /usr/src/new and out of the original directory into /usr/src/old for a time before removing them. Locally written commands that aren't distributed are kept in /usr/src/local and their binaries are kept in /usr/local. This allows /usr/bin, /usr/ucb, and /bin to correspond to the distribution tape (and to the manuals that people can buy). People wishing to use /usr/local commands are made aware that they aren't in the base manual. As manual updates incorporate these commands, they are moved to /usr/ucb.

A directory /usr/junk to throw garbage into, as well as binary directories /usr/old and /usr/new, are useful. The man command supports manual directories such as /usr/man/manj for junk and /usr/man/manl for local to make this or something similar practical.

4.8. Accounting for Connect Time and Process Resources

4.2/RT optionally records two kinds of accounting information: connect time accounting and process resource accounting. The connect time accounting information is stored in the file /usr/adm/wtmp, and is summarized by the program *ac*(8). The process time accounting information is stored in the file /usr/adm/acct, which is analyzed and summarized by the program *sa*(8).

If you need to charge for computing time, you can implement procedures based on the information provided by these commands. A convenient way to do this is to give commands to the clock daemon /etc/cron to be executed every day at a specified time. This is done by adding lines to /usr/lib/crontab; see *cron*(8) for details.

4.9. Controlling Resources

Resource control in the current version of 4.2/RT is elaborate compared to most UNIX operating systems. The disk quota facilities developed at the University of Melbourne have been incorporated in the system and allow control over the number of files and amount of disk space each user may use on each file system. In addition, the resources consumed by any single process can be limited by the mechanisms of *setrlimit*(2). As distributed, the latter mechanism is voluntary, though sites may choose to modify the login mechanism to impose limits not covered with disk quotas.

To use the disk quota facilities, the system must be configured with "options QUOTA". Then place file systems under the quota mechanism by creating a null file quotas at the root of the file system, running *quotacheck*(8), and modifying /etc/fstab to indicate the file system is read-write with disk quotas (an "rq" type field). Then run the program *quotaon*(8) to enable quotas.

Apply individual quotas using the quota editor *edquota*(8). Users may view their quotas (but not those of other users) with the *quota*(1) program. Use the *repquota*(8) program to summarize the quotas and current space usage on a particular file system or file systems.

You can enforce quotas with *soft* and *hard* limits. When a user first reaches a soft limit on a resource, a message appears on his/her terminal. If the user fails to lower the resource usage below the soft limit, the next *login* causes a warning about excessive usage. Should three login sessions go by with the soft limit breached, the system then treats the soft limit as a *hard* limit and disallows any allocations until enough space is reclaimed to bring the user back below the soft limit. Hard limits are strictly enforced, resulting in errors when a user tries to create or write a file. Each time a hard limit is exceeded the system will generate a message on the user's terminal.

Consult the document "Disc Quotas in a UNIX Environment" in Volume 2C of the *UNIX Programmer's Manual* and the related manual entries for more information.

4.10. Network Troubleshooting

If you have anything more than a trivial network configuration, from time to time you are bound to run into problems. Before blaming the software, first check your network connections. On networks such as the Ethernet, a loose cable tap or misplaced power cable can result in severely deteriorated service. The *netstat*(1) program may b aid in tracking down hardware malfunctions. In particular, look at the −i and −s options in the manual page.

Should you believe a communication protocol problem exists, consult the protocol specifications, and attempt to isolate the problem in a packet trace. The SO_DEBUG option may be supplied before establishing a connection on a socket, in which case the system will trace all traffic and internal actions (such as timers expiring) in a circular trace buffer. This buffer may then be printed out with the *trpt*(8C) program. Most servers distributed with the system accept a −d option that forces all sockets to be created with debugging turned on. Consult the appropriate manual pages for more information.

4.11. Monitoring Specific Files

As part of normal system operations, you should periodically review the following files (some of which are system-specific):

/etc/fstab	how disk partitions are used
/etc/disktab	disk partition sizes
/etc/printcap	printer data base
/etc/gettytab	terminal type definitions
/etc/remote	names and phone numbers of remote machines for *tip*(1)
/etc/group	group memberships
/etc/motd	message of the day
/etc/passwd	password file; each account has a line

/etc/rc.local	local system restart script; runs reboot; starts daemons
/etc/hosts	host name data base
/etc/networks	network name data base
/etc/services	network services data base
/etc/hosts.equiv	hosts under same administrative control
/etc/securetty	restricted list of ttys where root can log in
/etc/ttys	enables/disables ports
/etc/ttytype	terminal types connected to ports
/usr/lib/crontab	commands that are run periodically
/usr/lib/aliases	mail forwarding and distribution groups
/usr/adm/acct	raw process account data
/usr/adm/lpd-errs	line printer daemon error log
/usr/adm/messages	system error log
/usr/adm/ppd-errs	page printer error log
/usr/adm/shutdownlog	log of system reboots
/usr/adm/wtmp	login session accounting

5. ADVANCED INTERACTIVE EXECUTIVE (AIX) AND 4.2/RT CO-RESIDENCE

It is possible to have AIX and 4.2/RT systems on the same machine. Each system should have its own disk or disks. For the two systems to co-reside, the following steps must be taken (a two-disk system is described; a three-disk system is similar):

5.1. Installing AIX on an Existing 4.2/RT System

AIX normally expects to be booted from drive 0, so move the 4.2/RT system to drive 1 or drive 2. This can be done by copying the filesystems (with dump/restore) or by physically moving the disks. To move the disks, see *IBM RT PC User's Setup Guide*, 648967, and *IBM RT PC Options Installation*, 55X8838.

5.1.1. Creating a Minidisk (partition) Table on Existing 4.2/RT Disks

For the AIX installation procedure not to use the 4.2/RT disk (or disks), a minidisk (partition) table must be established on each 4.2/RT disk that uses all the available space. This is done by the *minidisk*(8R) utility, part of the *sautil*(8R) standalone utility. This procedure assumes there is no existing 4.2/RT minidisk (partition) table.

The procedure is:

(1) Boot up sautil utility. It is located in /usr/stand/sautil on an installed system, and on the standalone sautil diskette.

(2) Select the "minidisk" menu item.

(3) Initialize the minidisk directory - this makes all the space on the disk available.

(4) Create the standard partition tables by using the *standard* command (this corresponds to the normal a, b, and g partitions of a 4.2/RT disk.) It also creates a boot partition to hold the bootstrap, since this exists before the "a" partition.

5.2. Installing AIX and 4.2/RT on a New Machine

Install 4.2/RT first, then AIX as follows:

- create a 4.2/RT minidisk (partition) table as given below on each 4.2/RT disk
- install the 4.2/RT root and usr filesystems onto the first available disk (AIX uses drive 0 and possibly drive 1; 4.2/RT uses drives after AIX.)
- install AIX -- it should only use the drives set aside for it leaving the 4.2/RT drives alone because they have valid minidisk tables using all the available space.

5.3. Booting AIX

After AIX is installed, it puts its bootstrap into the boot block of drive zero. This means when the system is booted, AIX runs (since the default boot order is f0, f1, d0, d1, d2).

5.4. Booting 4.2/RT

There are two alternatives here. The simplest solution is to use a 4.2/RT boot diskette to boot 4.2/RT from drive 1. This requires manual intervention (or a non-standard boot diskette) since the the standard boot diskette attempts to boot hd(0,0)vmunix. Since the 4.2/RT root is on drive 1 (or drive 2), you should boot 4.2/RT from hd(1,0)vmunix or hd(2,0)vmunix as appropriate. A generic kernel then asks for the root disk (which is either hd1 or hd2 depending upon which drive is used for the root).

The other alternative is to change the boot order in non-volatile ram so the boot order is f0, f1, d1, d0, d2. This should be done in those cases where AIX is not used frequently. The "iplsource" option of *sautil*(8R) describes how to do this. In this case, a non-generic kernel must be used. See section 2.1.3 "Building New System Images" for information on building kernels. A generic kernel attempts to use hd0a as its root device and then panic as there is not a proper superblock there.

5.5. Creating a Minidisk Partition Table on a New 4.2/RT Disk

There are two reasons for creating a partition table on a 4.2 disk before the 4.2/RT installation:

- to prepare for eventual AIX installation (this is not critical as the partition table can always be added later)
- to use non-standard partitions. This is often done because the standard partitions do not fit every situation. In particular, it is often the case on small (40Mb) disks, the swap partition is insufficient to run large applications (such as window managers) and more swap space must be allocated. In other cases, it is desirable to create more partitions, or to have only a swap area and a large filesystem (on a second drive for example) rather than the standard three partitions.

(**WARNING**: changing the size or location of a partition containing a filesystem effectively destroys all the contents of that filesystem. You must do a dump/restore to change a filesystem's size and keep the contents). The procedure for creating a non-standard minidisk (partition) table is:

(1) Boot up *minidisk*(8R) utility (as described earlier).

(2) Initialize the minidisk directory by using the *initialization* command to make all the space available.

(3) Create standard partitions by using the *standard* command.

(4) Delete the partitions not needed (but keep the partition named "boot" as this is required to align the 4.2/RT minidisks on cylinder boundaries).

(5) Create (or recreate) the new partitions. Usually one makes the 'b' (swap) partition bigger, and the 'g' (usr) partition smaller.

5.6. Installing 4.2/RT on an Existing AIX Machine

As AIX automatically uses all the available disks, this is only feasible in two cases:

- a new disk can be added. In this case, create the 4.2/RT minidisk (partition) table and install 4.2/RT on the new disk.
- the AIX system must be dumped to tape or diskette. Then 4.2/RT is installed as described above and AIX is re-installed from the dumped tape or diskette.

5.7. Shared Swap Partitions

4.2/RT can use the AIX swap partition, but a non-generic kernel must be configured to do so. Assume AIX on drive 0, 4.2/RT on drive 1. This kernel could have a configuration like:

```
config  vmunix          root on hd1 swap on hd0 and hd1
```

5.8. Generating an Sautil Diskette

A bootable standalone diskette is generated by taking the program (such as */boot*, or */sys/standca/sautil.out*) and writing it onto a diskette with *doswrite* (see *dosread*(1)) with the appropriate options.

To generate a standalone bootable sautil diskette, do the following:

```
cd /sys/standca
make sautil.out
doswrite -i -b -v sautil.out
```

This page intentionally left blank.

Building 4.2/RT Systems with Config

ABSTRACT

This article is an updated version of an article entitled "Building 4.2BSD UNIX Systems with Config," written in June 1983 by Samuel J. Leffler and found in Volume 2C of the *UNIX Programmer's Manual.* The updates include additions and changes appropriate to the IBM RT PC. The article contains six chapters and four appendices:

1. **Introduction** describes *config* and its uses.

2. **Configuration File Contents** defines each element of a configuration file.

3. **System Building Process** describes the steps that build a bootable system image.

4. **Configuration File Syntax** describes the rules for writing a configuration file.

5. **Sample Configuration File** illustrates how to configure a sample IBM RT PC.

6. **Adding New System Software** describes some of the inner workings of the configuration process.

Appendix A. Configuration File Grammar is a compressed form of the actual *yacc*(1) grammar used by *config.*

Appendix B. Rules for Defaulting System Devices describes how *config* arrives at default values for device parameters.

Appendix C. Sample Configuration File lists the complete sample configuration file developed in Chapter 5.

Appendix D. Kernel Data Structure Sizing Rules describes the rules used at compile time and boot time to size certain system data structures.

1. INTRODUCTION

Config is a tool used in building 4.2/RT system images. It reads a file describing a system's tunable parameters and hardware support, and generates a collection of files used to build a copy of 4.2/RT appropriate to that configuration. *Config* simplifies system maintenance by isolating system dependencies in a single, easy-to-understand file.

This article describes how to use *config*(8) to configure and create bootable 4.2/RT system images.

Summary of Changes for the IBM RT PC

Significant changes to the original article are in the following sections:

- 4.1: Global Configuration Parameters
- 4.3: Device Specifications
- Appendix C: Sample Configuration File

2. CONFIGURATION FILE CONTENTS

A system configuration must include at least the following pieces of information:

- machine type
- cpu type
- system identification
- time zone
- maximum users
- location of the root file system
- available hardware

Config allows multiple system images to be generated from a single configuration description. Each system image is configured for identical hardware, but may have different locations for the root file system and, possibly, other system devices.

2.1. Machine Type

The *machine type* identifies the machine on which 4.2/RT will operate. The machine type is used to locate certain machine-specific data files and to select rules for constructing the configuration files.

2.2. Cpu Type

The *cpu type* identifies on which of possibly many cpus 4.2/RT will operate. Specifying more than one cpu type implies 4.2/RT should be configured to run on all the cpus specified. For those machines on which this is not possible, *config* prints a diagnostic message.

2.3. System Identification

The *system identification* is a name attached to the system, and often the machine on which the system is to run. The system identification is used to create a global C "#define" that in turn is used to isolate system-dependent code in the kernel.

The system identifier "GENERIC" is given to a system that will run on any cpu of a particular machine type; it should not otherwise be used for a system identifier.

2.4. Time Zone

The *timezone* in which the system will run affects the information returned by the *gettimeofday*(2) system call. This value is specified as the number of hours east or west of GMT. Negative numbers indicate a value east of GMT. The timezone specification may also indicate the type of daylight savings time rules to be applied.

2.5. Maximum Users

The system allocates system data structures at boot time based on the maximum users the system will support. This number, *maxusers*, is normally between 4 and 16, depending on the hardware and expected job mix. The rules used to calculate system data structures are discussed in Appendix D of this article.

2.6. Root File System Location

When the system boots it must know the location of the root of the file system tree. This location and the part(s) of the disk(s) to be used for paging and swapping must be specified to create a complete configuration description. You use the keyword *config* to specify these values. *Config* uses rules to calculate default locations for these items. These rules are described in Appendix B of this article.

When a generic system is configured, the root file system is left undefined until the system is booted. Therefore, the root file system need not be specified. You need only specify that the system is a generic system.

2.7. Hardware Devices

Part of the boot process is an *autoconfiguration* phase, when the system searches for those hardware devices the system builder has indicated might be present. This probing sequence requires certain pieces of information such as register addresses. A system's hardware may be configured with considerable flexibility or without any flexibility whatsoever. Most people do not configure hardware devices into the system unless:

- The devices are currently present on the machine
- The devices are due soon
- The devices are a safeguard against a hardware failure somewhere else at the site

  (Berkeley recommends configuring extra disks if an emergency requires moving one from a machine with hardware problems).

The bulk of the configuration file is usually devoted to hardware device specifications. Much of this article explains these specifications. Section 6.3 describes the autoconfiguration process for those planning to write new, or modify existing, device drivers.

2.8. Optional Items

In addition to the mandatory pieces of information described above, you can include various optional system facilities. For example, you can include support for monitoring disk quotas, and for tracing the performance of the virtual memory subsystem. You use the configuration file to specify any optional facilities to be configured into the system. The resultant files generated by *config* will automatically include the necessary pieces of the system.

3. SYSTEM BUILDING PROCESS

This section describes the steps necessary to build a bootable system image. We assume the system source is located in the /sys directory and that, initially, the system is being configured from source code.

Under normal circumstances there are five steps in building a system:

(1) Create a configuration file for the system.

(2) Make a directory in which to construct the system.

(3) Run *config* on the configuration file to generate the files required for compiling and loading the system image.

(4) Construct the source code dependency rules for the configured system.

(5) Compile and load the system with *make*(1).

Steps 1 and 2 are usually done only once. When a system configuration changes, you usually just run *config* on the modified configuration file, rebuild the source code dependencies, and remake the system. Sometimes, however, configuration dependencies may not be noticed. Then it is necessary to clean out the relocatable object files saved in the system's directory. This is discussed later.

3.1. Creating a Configuration File

Configuration files normally reside in the /sys/conf directory. It is easiest to construct a configuration file by copying an existing configuration file and modifying it. This distribution includes a sample configuration file.

The configuration file must have the same name as the directory in which the configured system is to be built. Further, *config* assumes this directory is located in the parent directory of the directory in which *config* is run. For example, the generic system has a configuration file named /sys/conf/GENERIC, and an accompanying directory named /sys/GENERIC. In general it is unwise to move your configuration directories out of /sys as most of the system code and the files created by *config* use pathnames of the ../ form. If you are running out of space on the file system where the configuration directories are located, there is a mechanism for sharing relocatable object files between systems. This is described later.

When building your configuration file, be sure to include the items described in Chapter 2. In particular, you must specify machine type, cpu type, time zone, system identifier, maximum users, and root device. Specifying the hardware present may take a bit of work, particularly if your hardware is configured at non-standard places (e.g. device registers located at unexpected places, or devices not supported by the system). Chapters 4, 5, and 6 of this article should be of help to you. If the devices to be configured are not described in the sample configuration file, you should check the manual pages in Volume I, Section 4, of this manual or Volume I, Section 4, of the *UNIX Programmer's Manual.* For each supported device, the manual page synopsis entry gives a sample configuration line.

Once the configuration file is complete, run it through *config* and look for any errors. Don't try to use a system that *config* has complained about; the results are unpredictable. For the most part, *config*'s error diagnostics are self explanatory; sometimes the line numbers given with the error messages are off by one.

A successful run of *config* on your configuration file will generate several files in the configuration directory. These files are:

- A file to be used by *make*(1) in compiling and loading the system
- One file for each possible system image for your machine, which describes where swapping areas, the root file system, and other miscellaneous system devices are

located

- A collection of header files, one per possible device the system supports, which defines the hardware configured
- A file containing the I/O configuration tables used by the system during its *autoconfiguration* phase

Unless you have reason to doubt *config* or are curious how the system's autoconfiguration scheme works, you should never have to look at any of these files.

3.2. Constructing Source Code Dependencies

When *config* finishes generating the files needed to compile and link your system, it terminates with a message of the form:

Don't forget to run make depend.

This message is a reminder that you should change to the configuration directory for the system just configured and type:

make depend

This step builds the rules used by *make* to recognize interdependencies in the system source code, and insures that any changes to system source code will result in the proper modules being recompiled the next time *make* is run.

This step is particularly important if your site makes changes to the system include files. The rules specify which source code files are dependent on which include files. Without these rules, *make* will not recognize when it must rebuild modules because a system header file has been modified. Note that dependency rules created by this step only reflect directly included files. That is, if file A includes a file B, which includes a file C, and then C is modified, *make* will not recognize that A should be recompiled. Therefore, it is best to keep include file dependencies only one level deep.

3.3. Building the System

The makefile constructed by *config* allows a new system to be rebuilt by simply typing:

make image-name

For example, if you have named your bootable system image "vmunix", then **make vmunix** will generate a bootable image named "vmunix". You use a different system image name if the root file system location and/or swapping configuration differ from those in the bootable system image. The makefile that *config* creates has entry points for each system image defined in the configuration file. Thus, if you have configured "vmunix" to be a system with the root file system on "hd0" and "hd1vmunix" to be a system with the root file system on "hd1," then **make vmunix hd1vmunix** will generate binary images for each.

Note that the name of a bootable image is different from the system identifier. All bootable images are configured for the same system; only the information about the root file system and paging devices differ. (This is described in more detail in Chapter 4.)

The last step in the system building process is to rearrange certain commonly used symbols in the system image symbol table. The makefile generated by *config* does this automatically for you. This approach is advantageous for programs such as *ps*(1) and *vmstat*(1), that run much faster when the symbols they need are located at the front of the symbol table. Remember also that many programs expect the currently executing system to be named /vmunix. If you install a new system and name it something other than /vmunix, many programs are likely to give strange results.

If you are making a kernel that contains the debugger, the makefile target is:

make image_name.ws

For example, if the bootable image would normally be vmunix, then vmunix.ws is the kernel with the debugger and should be specified on the *make* command line. It should be installed as /vmunix, of course.

3.4. Sharing Object Modules

If you have many systems that are all built on a single machine there are at least two approaches to saving time in building system images. The best way is to have a single system which is run on all machines. This is attractive since it minimizes disk space used and time required to rebuild systems after making changes. However, it is often true that one or more systems will require a separately configured system image. This may be because of limited memory (building a system with many unused device drivers can be expensive), or configuration requirements (one machine may be a development machine where disk quotas are not needed, while another is a production machine where they are). In these cases it is possible for common systems to share relocatable object modules that are not configuration-dependent. Most of the modules in the directory /sys/sys are of this sort.

To share object modules across systems, you should first build a generic system. Then for each system, configure the system as before, but before recompiling and linking the system, type:

make links

This step searches the system for source modules that are safe to share between systems, and generates symbolic links in the current directory to the appropriate object modules in the ../GENERIC directory. This request also generates a shell script, "makelinks", which you may want to check for accuracy. The file /sys/conf/defines contains a list of symbols that Berkeley feels are safe to ignore when checking the source code for modules to be shared. Note that this list includes the definitions used to compile in the virtual memory tracing facilities and the trace point support used only rarely (even at Berkeley). It may be necessary to modify this file to reflect local needs. Note further that, as already mentioned, interdependencies that are not directly visible in the source code are not caught. This means that if you place per-system dependencies in an include file, they will not be recognized and the shared code may be selected in an unexpected fashion.

3.5. Building Profiled Systems

It is simple to configure a system that automatically collects profiling information as it operates. The profiling data can be collected with *kgmon*(8) and processed with *gprof*(1) to obtain information regarding the system's operation. Profiled systems maintain histograms of the program counter as well as the number of invocations of each routine. The *gprof*(1) command also generates a dynamic call graph of the executing system, and propagates time spent in each routine along the arcs of the call graph. (Consult the gprof documentation for more information.) The program counter sampling can be driven by the system clock or a real time clock (if you have one). The latter is highly recommended; using the system clock results in statistical anomalies, and time spent in the clock routine will not be accounted for correctly.

To configure a profiled system, the **−p** option should be supplied to *config*. A profiled system is about 5-10% larger in its text space because of the calls to count the subroutine invocations. When the system executes, the profiling data is stored in a buffer that is 1.2 times the size of the text space. The overhead for running a profiled system varies; under normal load Berkeley sees 5-25% of the system time spent in the profiling code.

Note that systems configured for profiling should not be shared as described above unless all the other shared systems are also to be profiled.

3.6. Building a System with pcc

In building a system with *pcc* rather than *hc*, be aware of two problems caused by the larger kernels *pcc* generates. The first problem is caused by the kernel debugger becoming larger than the kernel assumes an *hc*-compiled debugger will be. (The assumption is set in the file /sys/ca/rdb.h.) If the debugger exceeds that size, the make of rdb.ws will fail, generating an error message advising you to increase the value of **RDB_END**.

The second problem is that kernels having several options and pseudo devices defined may become too large to boot, when generated by *pcc*. This will be true if the total kernel size -- the size of the kernel bss plus the total boot size -- is larger than 1 Mb.

4. CONFIGURATION FILE SYNTAX

This section describes the specific rules used in writing a configuration file. A complete grammar for the input language is in Appendix A, and may be useful for resolving syntax errors.

A configuration file contains three logical pieces of information:

- parameters global to all system images
- parameters specific to each system image to be generated
- device specifications

4.1. Global Configuration Parameters

The global configuration parameters are machine type, cpu types, options, time zone, system identifier, and maximum users. Each is specified with a separate line in the configuration file.

machine *type*

The system runs on the machine type specified. No more than one machine type can appear in the configuration file. For the IBM RT PC, **ca** is the legal value.

cpu *"type"*

This system runs on the cpu type specified. More than one cpu type specification can appear in a configuration file. **"IBMRTPC"** is the legal value on the IBM RT PC. Note that the quotation marks are required.

options *optionlist*

The listed optional code is compiled into the system. Options in this list are separated by commas. Possible options are listed at the top of the generic makefile. A line of the form "options DEBUG" generates a define of the form −DDEBUG in the resulting makefile. A line of the form "options ROROOTDEV=0x0200" generates a line of the form −DROROOTDEV=0x0200. An option may be given a value by following its name with "=" and the value enclosed in (double) quotes. None of the standard options use such a value. The available options appear in the following table.

Option	Effect
DEBUG	Compiles various debugging code into the kernel
SHOW_LOAD	Shows load average in front panel LED displays
QUOTA	Compiles code for disk quotas
INET	Compiles code for network support
LF_DELAY=n	Specifies a line-feed delay for console monochrome output, making kernel debugging messages easier to read
LP_LOG=n	If n! = 0 then log console messages on the printer
GPROF	Includes kernel profiling code

Option	Effect
CLOCKDEBUG	Includes clock debugging code
IODEBUG	Includes SLIH tracing IO debugging code
SYSCALLTRACE	Traces system calls and their arguments
RDB	Specifies that the kernel will interface with the debugger
AEDDEBUG	Includes debugging code for the experimental display
PGINPROF	Profiles VM usage
ROROOTDEV = 0xmmnn	Causes the root disk to be mounted read-only if it's major/minor
XWM	Must be defined to use a pseudo-device xemul
SECURE	Allows *kbdlock*(1) to lock the console keyboard
DUALCALL	Allows running a.outs with either calling sequence

Additional options associated with certain peripheral devices are listed in the Synopsis section of the manual page for the devices.

timezone *number* [**dst** [*number*]]

This specifies the timezone you are in: the number of hours your time zone is west of GMT. EST is five hours west of GMT; PST, eight. Negative numbers indicate hours east of GMT. If you specify **dst**, the system will operate under daylight savings time. An optional integer or floating point number can be included to specify a particular daylight saving time correction algorithm. The default value is 1, for the United States. Other values are: 2 (Australia), 3 (Western Europe), 4 (Middle Europe), and 5 (Eastern Europe). See *gettimeofday*(2) and *ctime*(3) for more information.

ident *name*

This system is known as *name*. The sample configuration file uses the name **SAMPLE**.

maxusers *number*

This is the maximum number of simultaneously active users expected on the system, and is used to size several system data structures. On the IBM RT PC, the minimum value for maxusers is 4.

4.2. System Image Parameters

Multiple bootable images may be specified in a single configuration file. The systems will have the same global configuration parameters and devices, but the location of the root file system and other system specific devices may be different. You specify a system image using a "config" line:

config *sysname config-clauses*

The *sysname* field is the name given to the loaded system image; almost everyone names their standard system image "vmunix". The configuration clauses are one or more specifications showing where the root file system is located, how many paging devices there are, and where they go. The device used by the system to process argument lists during *execve*(2) calls can also be specified, though in practice this is almost always done by *config* using one of its rules for selecting default locations for system devices.

A configuration clause is one of the following

root [**on**] *root-device*
swap [**on**] *swap-device* [**and** *swap-device*]
dumps [**on**] *dump-device*
args [**on**] *arg-device*

(The "on" is optional.) Multiple configuration clauses are separated by white space; *config* allows specifications to be continued across multiple lines by beginning the continuation line with a tab character. The "root" clause specifies where the root file system is located, the "swap" clause specifies swapping and paging area(s), the "dumps" clause can be used to force system dumps to a particular device, and the "args" clause can be used to force argument list processing for *execve* to a particular disk.

The device names supplied in the clauses may be fully specified as a device, unit, and file system partition; or underspecified, in which case *config* will use its own rules to select default unit numbers and file system partitions. The defaulting rules are a bit complicated, as they are dependent on the overall system configuration. For example, the swap area need not be specified at all if the root device is specified; the swap area is placed in the "b" partition of the disk where the root file system resides. Appendix B contains a complete list of the defaulting rules used in selecting system configuration devices.

The device names are translated to the appropriate major and minor device numbers on a per-machine basis. A file, /sys/conf/devices.machine (where "machine" is the machine type specified in the configuration file), is used to map a device name to its major block device number. The minor device number is calculated using the standard disk partitioning rules: on unit 0, partition "a" is minor device 0, partition "b" is minor device 1, and so on; for units other than 0, add 8 times the unit number to get the minor device.

If the default mapping of device name to major/minor device number is incorrect for your configuration, it can be replaced by an explicit specification of the major/minor device. You do this by substituting

major *x* **minor** *y*

where the device name would normally be found.

Normally, the areas configured for swap space are sized by the system at boot time. If a non-standard partition size is to be used for one or more swap areas, you can add a "size" specification to the device name for the swap area. For example,

config vmunix root on hd0 swap on hd0b size 1200

would force swapping to be done in partition "b" of "hd0" and the swap partition size would be set to 1200 sectors. A swap area sized larger than the associated disk partition is trimmed to the partition size.

To create a generic configuration, only the clause "swap generic" should be specified; any extra clauses will cause an error.

4.3. Device Specifications

You must specify to *config* each device attached to a machine, so that the generated system will know to probe for it during the autoconfiguration process at boot time. Hardware specified in the configuration file need not actually be present on the machine where the generated system is run. Only the hardware actually found at boot time will be used by the system.

The specification of hardware devices in the configuration file parallels the interconnection hierarchy of the machine to be configured. A configuration file must identify what adapters are present. A device description can provide a complete definition of the possible configuration parameters, or can leave certain parameters undefined; at boot time the system probes for these missing values. The latter approach allows a single device configuration list to match many possible physical configurations. This approach, termed *wildcarding*, provides more flexibility in the physical configuration of a system. If a disk must be moved for some reason, the system will still locate it at the alternate location.

For the IBM RT PC, a device specification takes one of the following forms:

controller *device-name device-info*
device *device-name device-info*
disk *device-name device-info*
tape *device-name device-info*

A *controller is an adapter that controls one or more disks or tapes; everything else is a device.*

The *device-name* is one of the standard device names, concatenated with the *logical* unit number assigned to the device. (The *logical* unit number may differ from the *physical* unit number shown on the front of a device like a disk; the *logical* unit number refers to the 4.2/RT device, not the physical unit number). Standard device names for the IBM RT PC are documented in Volume I,, Section 4, of this manual.

The *device-info* clause specifies how the hardware is connected in the interconnection hierarchy. The beginning of the hierarchy is defined as follows:

controller iocc0 **at nexus** ?

The remaining legal interconnections are:

- A controller can be connected to another controller
- A disk or tape is always attached to a controller
- Devices are always attached to controllers

On the IBM RT PC, the controller specification takes the form:

controller xxcn **at** iocc0 **csr** addr **priority** irq

where

"xx"	is the two- or three-letter name of the device
"n"	is the controller number (0, 1, 2, 3, etc.)
"addr"	is the controller adapter base address
"irq"	is the PC/AT IRQ level for the adapter

For example, the line:

controller hdc0 **at** iocc0 csr 0xf00001f0 priority 14

specifies that the hard disk controller (adapter) is at iocc0 (required); its csr address is 0xf00001f0; and its device interrupt request level is 14. Note that interrupt service routines are not specified here. Each adapter has only one interrupt service routine, specified in the iocc_driver structure within the device driver.

For a slave device on the IBM RT PC, the specification takes one of the following forms:

disk xxn **at** xxcy drive z
tape xxn **at** xxcy drive z

where

"xx"	is the two- or three-letter name of the device
"n"	is the unit number
"y"	is the controller number
"z"	is the slave unit number

For example, the following is a specification for a hard disk:

disk hd0 **at** hdc0 drive 0

For a non-slave device on the IBM RT PC, the specification takes the form:

device xxn **at** iocc0 csr addr priority irq

where

"xx"	is the two- or three-letter name of the device
"n"	is the adapter unit number (0, 1, 2, 3, etc.)
"addr"	is the device adapter base address
"irq"	is the PC/AT IRQ level for the adapter

For example, the following is a specification for the four-line serial card:

device asy0 **at** iocc0 csr 0xf0001230 priority 9

Any piece of hardware that can be connected to a specific controller can also be wildcarded across multiple controllers, by specifying the controller number as "?".

The final piece of information needed by the system to configure devices is some indication of where or how a device will interrupt. On the IBM RT PC, interrupt routines are specified in the driver rather than in the configuration file.

Certain device drivers require extra information passed to them at boot time to tailor their operation to the actual hardware present. The drivers for the terminal multiplexors need to know which lines are attached to modem lines so that no one will be allowed to use them unless a connection is present. Therefore, one last parameter may be specified for a device, a *flags* field. It has the syntax

flags *number*

and is usually placed after the *csr* specification. The *number* is passed directly to the associated driver. You should consult the manual pages in Volume I, Section 4, of this manual to determine how each driver uses this value (if at all). Communications interface drivers commonly use the flags to indicate whether modem control signals are in use.

The exact syntax for each specific device is given in the Synopsis section of its manual page in Volume I, Section 4, of this manual.

4.4. Pseudo-Devices

Some drivers and software subsystems are treated like device drivers without any associated hardware. To include any of these pieces, a "pseudo-device" specification must be used. A specification for a pseudo-device takes the form

pseudo-device *device-name* [*howmany*]

Examples of pseudo-devices are **bk**, the Berknet line discipline; **pty**, the pseudo terminal driver (where the optional *howmany* value indicates the number of pseudo terminals to configure, with 32 as the default); and **inet**, the DARPA Internet protocols (one must also specify INET in the "options" statement). Other pseudo-devices for the network include **loop**, the software loopback interface; **imp** (required when a CSS or ACC imp is configured); and **ether** (used by the Address Resolution Protocol on 10 Mb/sec Ethernets). More information on configuring each of these can also be found in Section 4 of the *UNIX Programmer's Manual.*

Other pseudo-devices specific to the IBM RT PC are **ms** for the mouse, **mono** for the monochrome display, **aed** for the IBM Academic Information Systems experimental display, **apasixteen** for the IBM 6155 Extended Monochrome Graphics Display, **apaeightc** for the IBM 6154 Advanced Color Graphics Display, **apaeight** for the IBM 6153 Advanced Monochrome Graphics Display, **xemul** for the X support, and **ap** for the IBM 3812 Pageprinter.

5. SAMPLE CONFIGURATION FILES

This chapter illustrates how to configure a sample IBM 6150 Model 25 (floor model) that will run in a networking environment.

5.1. Floor Model

The following table lists the hardware to be configured.

Item	Connection	Name	Reference
cpu	"IBMRTPC"		
controller	nexus ?	iocc0	
U-B Ethernet card	iocc0	un0	un(4)
disk controller	iocc0	hdc0	hd(4)
disk controller	iocc0	hdc1	
diskette controller	iocc0	fdc0	fd(4)
disk	hdc0	hd0	hd(4)
disk	hdc0	hd1	
disk	hdc1	hd2	
async controller	iocc0	asy0	asy(4)
async controller	iocc0	asy1	asy(4)
async controller	iocc0	asy4	asy(4)
async controller	iocc0	psp0	psp(4)
experimental display	iocc0	aed	ibmaed(4)
advanced monochrome display	iocc0	apa8	ibm6153(4)
extended monochrome display	iocc0	apa16	ibm6155(4)
monochrome display	iocc0	mono	ibm5151(4)
printer controller	iocc0	lp0	lp(4)
diskette	fdc0	fd0	fd(4)
tape controller	iocc0	stc0	st(4)
tape	stc0	st0	st(4)

The following steps illustrate how to build the configuration file for this system.

(1) Fill in the global configuration parameters.

The machine is an IBM RT PC, with a *machine type* of **ca**. This system is to run only on this one processor; the *cpu type* is **"IBMRTPC"**. We will use the INET option, because we plan to use the DARPA standard Internet protocols. The system identifier is **SAMPLE**. The maximum users we plan to support is about 16. Thus the beginning of the configuration file looks like this:

```
#
# Sample Configuration File for the IBM RT PC
#
machine         ca
cpu             "IBMRTPC"
ident           SAMPLE
timezone        8 dst
maxusers        16
options         INET
```

(2) Add the specification for a single system image.

Our standard system has the root on "hd0" and swapping on both "hd0" and "hd1".

```
config          vmunix          root on hd0 swap on hd0 and hd1
```

(3) Specify the hardware.

Transcribe the information from the preceding table:

```
controller  iocc0   at nexus ?
device      un0     at iocc0 csr 0xf4080000 priority 6
controller  hdc0    at iocc0 csr 0xf00001f0 priority 14
controller  hdc1    at iocc0 csr 0xf0000170 priority 14
controller  fdc0    at iocc0 csr 0xf00003f2 priority 6
disk        hd0     at hdc0 drive 0
disk        hd1     at hdc0 drive 1
disk        hd2     at hdc1 drive 0
device      asy0    at iocc0 csr 0xf0001230 priority 9 flags 0x0f
device      asy1    at iocc0 csr 0xf0002230 priority 10
device      asy4    at iocc0 csr 0xf00003f8 priority 4
device      lp0     at iocc0 csr 0xf00003bc priority 7
device      fd0     at fdc0 drive 0
controller  stc0    at iocc0 csr 0xf00001e8 priority 12
tape        st0     at stc0 drive 0
device      psp0    at iocc0 csr 0xf0008000 priority 2 flags 0x03
```

(4) Add the required pseudo-devices:

```
pseudo-device       mono
pseudo-device       pty
pseudo-device       loop
pseudo-device       inet
pseudo-device       ether
pseudo-device       ms
pseudo-device       aed
pseudo-device       apaeight
pseudo-device       apaeightc
pseudo-device       apasixteen
```

This completes the sample configuration file. It appears in its entirety in Appendix C.

5.2. Miscellaneous Comments

Note that the sample system does not use either disk quotas or 4.1BSD compatibility mode. To use these optional facilities or others, Berkeley recommends cleaning out your current configuration, reconfiguring the system, then recompiling and relinking the system image(s). You could avoid this, of course, by figuring out which relocatable object files are affected by the reconfiguration, and then reconfiguring and recompiling only the affected files. Use this technique carefully.

6. ADDING NEW SYSTEM SOFTWARE

This chapter is not for the novice. It has four sections:

- how to modify system code
- how to add a device driver to 4.2/RT
- how device drivers are autoconfigured under 4.2/RT
- how to add non-standard system facilities to 4.2/RT

6.1. How to Modify System Code

To make site-specific modifications to the system, it is best to bracket them with

```
#ifdef SITENAME
...
#endif
```

This allows your source to be distributed to others easily, and also simplifies *diff*(1) listings. If you choose not to use a source code control system (e.g. SCCS, RCS), and perhaps even if you do, it is recommended that you save the old code with something of the form:

```
#ifndef SITENAME
...
#endif
```

Berkeley tries to isolate site-dependent code in individual files that may be configured with pseudo-device specifications.

Identify machine-specific code with "#ifdef ca".

6.2. How to Add Device Drivers to 4.2/RT

The I/O system and *config* have been designed so you can add new device support easily. The system source directories are organized as follows:

/sys/h	machine independent include files
/sys/sys	machine independent system source files
/sys/conf	site configuration files and basic templates
/sys/net	network independent, but network related code
/sys/netinet	DARPA Internet code
/sys/ca	IBM RT PC specific mainline code
/sys/caif	IBM RT PC network interface code
/sys/caio	IBM RT PC device drivers and related code
/sys/cacons	IBM RT PC console device drivers and related code

Existing block and character device drivers for the IBM RT PC reside in "/sys/ca" and "/sys/caio". Network interface drivers reside in "/sys/caif". Any new device drivers should be placed in the appropriate source code directory and named so as not to conflict with existing devices. Normally, definitions for things like device registers are placed in a separate file in the same directory. For example, "psp.c" is the name of the "psp" device driver, and "pspreg.h" is the name of its associated include file.

Once the source for the device driver has been placed in a directory, you should modify /sys/conf/files.machine and, possibly, /sys/conf/devices.machine. The two "files" files in the conf directory contain a line for each source or binary-only file in the system. Machine-independent files are located in /sys/conf/files, while machine-specific files are in /sys/conf/files.machine. The devices.machine file is used to map device names to major block device numbers. If the device driver being added provides support for a new disk, you will want to modify this file. (The format is obvious.) Note that the *.machine* suffix refers to the specific machine on which you're working. On the IBM RT PC, *.machine* becomes *.ca.*

The format of these two files has grown somewhat complex over time. Entries are normally of the form:

caio/foo.c **optional** foo **device-driver**

where the keyword *optional* indicates that to compile the "foo" driver into the system, it must be specified in the configuration file. If instead the driver is specified as *standard,* the file will be loaded no matter what configuration is requested. This is not normally done with device drivers.

Aside from including the driver in the appropriate "files" file, it must also be added to the device configuration tables. These are located in the /sys/ca/conf.c file. If you don't understand what to add to this file, you should study an entry for an existing driver. Remember that the position in the block device table specifies what the major block device driver number is; this number is needed in the "devices.machine" files if the device is a disk.

With the configuration information in place, your configuration file appropriately modified, and a system reconfigured and rebooted, you should incorporate the shell commands needed to install the special files in the file system to the /dev/MAKEDEV or /dev/MAKEDEV.local file. This is discussed in the article "Operating Academic Information Systems 4.2."

6.3. Autoconfiguration on the IBM RT PC

4.2/RT requires all device drivers to conform to a set of rules that allow the system to:

(1) Support system configuration at boot time, and

(2) Manage resources so as not to crash when devices request unavailable resources.

The IBM RT PC I/O control channel (IOCC) uses a set of translation control word (TCW) registers to convert from the PC/AT I/O bus address space into the IBM RT PC address space when using direct memory access (DMA). There is a structure of type *struct iocc_hd* in the system per DMA channel used to manage these resources. This structure also contains a linked list where devices waiting for resources to complete DMA activity have requests waiting.

There are three central structures used to write drivers for controllers:

struct iocc_ctlr -- the controller structure

struct iocc_driver -- the driver structure

struct iocc_device -- the device structure

These are defined in the . . . /caio/ioccvar.h file.

The elements are analogous to the VAX structures, except for the following:

ic_irq	is the irq level specified as 'priority' for a controller
iod_irq	is the irq level specified as 'priority' for a device
idr_intr	specifies the interrupt service routine for this driver

idr_csr	is the offset to a read/write register that can be interrogated for the existence of the device. Values at addr[csr + 0] and addr[csr + 1] will be tested for a non-existent device (compared to the contents of a "standard" non-existent device).

Devices that do not do DMA I/O can often use only two of these structures (iocc_driver and iocc_device). Each *controller* specified in the config file has an associated line in *struct iocc_ctlr iocc_cinit[]* and each device or slave has an entry in *struct iocc_device iocc_dinit[]* generated automatically in ioconf.c in the config directory. The *iocc_ctlr* and *iocc_device* structures are in one-to-one correspondence with the definitions of controllers and devices in the system configuration. Each driver has a *struct iocc_driver* structure specifying an internal interface to the rest of the system.

The specification:

```
controller hdc0 at iocc0 csr 0xf00001f0
```

would cause a *struct iocc_ctlr* to be declared and initialized in the file *ioconf.c* for the system configured from this description. Similarly specifying:

```
disk hd0 at hdc0 drive 0
```

would declare a related *iocc_device* in the same file. The *hd.c* driver which implements this driver specifies in declarations:

```
int hdprobe(), hdslave(), hdattach(), hdint();
int hdwstart, hdwatch();            /* watch routine */

/* hddinfo contains pointers to the slaves (drives) */
struct iocc_device *hddinfo[NHD];

struct iocc_ctlr *hdminfo[NHDC];


struct iocc_driver hdcdriver = {
        hdprobe, hdslave, hdattach,
/*  dgo     addr    dname  dinfo   mname minfo  intr    csr      */
        0, hdstd, "hd", hddinfo, "hdc", hdminfo, hdint, 2
```

which initializes the *iocc_driver* structure. The driver will support some number of controllers named *hdc0*, *hdc1*, etc, and some number of drives named *hd0*, *hd1*, etc. where the drives may be on any of the controllers (that is, there is a single linear name space for devices, separate from the controllers.)

We now explain the fields in the various structures. It may help to look at a copy of *caio/ioccreg.h*, *caio/ioccvar.h* and drivers such as *hd.c* and *asy.c* while reading the descriptions of the various structure fields.

6.3.1.1. iocc_driver structure

One of these structures exists per driver. It is initialized in the driver and contains functions used by the configuration program and by the DMA resource routines. The fields of the structure are:

idr_probe

The probe routine is given the adapter base address and should return one of the following values:

PROBE_BAD (0)	The device is bad, didn't really exist, etc.
PROBE_NOINT (1)	The device is there, but either cannot interrupt easily or the driver isn't smart enough to do so. Use the irq from the configuration information.
PROBE_OK (2)	The device is there, is healthy, and should have interrupted. If it did not actually interrupt, then a message will be printed and the device ignored.

The PROBE_DELAY(n) macro can be used when waiting for an interrupt to happen in the probe routine. It behaves exactly like DELAY(n), which delays for *n* microseconds, but will return as soon as an interrupt has happened.

idr_slave
This routine is called with a *iocc_device* structure (yet to be described) and the address of the device controller. It should determine whether a particular slave device of a controller is present, returning 1 if it is and 0 if it is not.

idr_attach
The attach routine is called after the autoconfigure code and the driver concur that a peripheral exists attached to a controller. This is the routine where internal driver state about the peripheral can be initialized.

The attach routine performs a number of functions. The first time any drive is attached to the controller it starts the timeout routine which watches the disk drives to make sure that interrupts aren't lost. It also initializes, for devices which have been assigned *iostat* numbers (when iod->iod_dk >= 0), the transfer rate of the device in the array *dk_mspw*, the fraction of a second it takes to transfer a 16-bit word. It increments the count of the number of devices on this controller, so that search commands can later be avoided if the count is exactly 1.

idr_dgo
Is the routine which is called by the DMA resource management routines when an operation is ready to be started (because the required resources have been allocated). It is not used by programmed I/O routines, such as hd.c.

idr_addr
Are the conventional addresses for the device control registers. This information is used by the system to look for instances of the device supported by the driver. When the system probes for the device it first checks for a control-status register located at the address indicated in the configuration file (if supplied), then uses the list of conventional addresses pointed to be *idr_addr*.

idr_dname
Is the name of a *device* supported by this controller; thus the disks on a fixed disk controller are called *hd0*, *hd1*, etc. That is because this field contains *hd*.

idr_dinfo
Is an array of back pointers to the *iocc_device* structures for each device attached to the controller. Each driver defines a set of controllers and a set of devices. The device address space is always one-dimensional, so that the presence of extra controllers may be masked away (e.g. by pattern matching) to take advantage of hardware redundancy. This field is filled in by the configuration program, and used by the driver.

idr_mname
The name of a controller, e.g. *hdc* for the *hd.c* driver. The first controller is called *hdc0*, etc.

idr_minfo
The backpointer array to the structures for the controllers.

idr_intr
The interrupt routine is called after the device receives an interrupt. It returns 0 if the interrupt really was for that device. Otherwise, it returns 1 so that multiple devices may share the same interrupt level.

idr_csr
The offset from idr_addr that PROBE uses to see if a device exists; idr_csr must be the offset to a read-write location that is safe for PROBE to read-write during autoconfig.

6.3.1.2. iocc_ctlr structure

One of these structures per controller exists. The fields link the controller to its adapter and contain the state information about the devices on the controller. The fields are:

ic_driver
A pointer to the *struct iocc_driver* for this driver, which has fields as defined above.

ic_ctlr
The controller number for this controller, e.g. the 0 in *shdc0.*

ic_alive
Set to 1 if the controller is considered alive; currently, always set for any structure encountered during normal operation. That is, the driver will have a handle on a *iocc_ctlr* structure only if the configuration routines set this field to a 1 and entered it into the driver tables.

ic_tab
This buffer structure is a place where the driver hangs the device structures which are ready to transfer. Each driver allocates a buf structure for each device (e.g. *hddtab* in the *hd.c* driver) for this purpose. You can think of this structure as a device-control-block, and the buf structures linked to it as the unit-control-blocks. The code for dealing with this structure is stylized; see the *fd.c* or *hd.c* driver for the details. If the *dmago* routine is to be used, the structure attached to this *buf* structure must be:

- A chain of *buf* structures for each waiting device on this controller.
- On each waiting *buf* structure another *buf* structure which is the one containing the parameters of the I/O operation.

ic_addr
Address of the device in I/O space.

ic_irq The interrupt request level for this device.

ic_channel
The device's dma channel (dma devices only).

ic_party
The device's dma party type. It must be set to DM_THIRDPARTY (dma devices only).

ic_transfer
Transfer flags for dma (dma devices only).

ic_cmd
Not used

6.3.1.3. iocc_device structure

One of these structures exist for each device. Devices which are not attached to controllers or which perform no DMA I/O may have only a device structure. Thus *asy* and *lp* devices have only *iocc_device* structures. The fields are:

iod_driver
A pointer to the *struct iocc_driver* structure for this device type.

iod_unit
The unit number of this device, e.g. 0 in **hd0**, or 1 in **asy1**.

iod_ctlr
The number of the controller on which this device is attached, or −1 if this device is not on a controller.

iod_slave
The slave number of this device on the controller which it is attached to, or −1 if the device is not a slave. Thus a disk which was unit 0 on a disk adapter would have *iod_slave* 0. It might or might not be *hd0*, that depends on the system configuration specification.

iod_irq
The interrupt request level for this device.

iod_addr
The control-status register address of this device.

iod_dk
The iostat number assigned to this device. Numbers are assigned to disks only, and are small positive integers which index the various *dk_** arrays in *<sys/dk.h>*.

iod_flags
The optional "flags *xxx*" parameter from the configuration specification was copied to this field, to be interpreted by the driver. If *flags* was not specified, then this field will contain a 0.

iod_alive
The device is really there. Presently set to 1 when a device is determined to be alive, and left 1.

iod_type
The device type, to be used by the driver internally.

iod_physaddr
The physical memory address of the device control-status register. This is used in the device dump routines typically.

iod_mi
A *struct iocc_ctlr* pointer to the controller (if any) on which this device resides.

iod_hd
A *struct iocc_hd* pointer to the DMA channel this device uses.

6.3.1.4. DMA Resource Management Routines

DMA drivers are supported by a collection of utility routines which manage DMA resources. If a driver attempts to bypass the DMA routines, other drivers may not operate properly. The major routines are: *dma_setup* to allocate DMA resources, *dma_done* to release previously allocated resources, and *dma_go* to initiate DMA.

If the presentation here does not answer all the questions you may have, consult the file /sys/caio/dma.c

6.3.2. Autoconfiguration Requirements

Basically all you have to do is write a *idr_probe* and a *idr_attach* routine for the controller. It suffices to have a *idr_probe* routine which just returns PROBE_NOINT, and a *idr_attach* routine which does nothing. Making the device fully configurable requires, of course, more work, but is worth it if you expect the device to be in common usage and want to share it with others.

If you managed to create all the needed hooks, then make sure you include the necessary header files; the ones included by *caio/lp.c* are nearly minimal. Order is important here, don't be surprised at undefined structure complaints if you order the includes wrongly. Finally if you get the device configured in, you can try bootstrapping and see if configuration messages print out about your device. It is a good idea to have some messages in the probe routine so that you can see that you are getting called and what is going on. If you do not get called, then you probably have the control-status register address wrong in your system configuration. The autoconfigure code notices that the device doesn't exist in this case and you will never get called.

Assuming that your probe routine works and you manage to generate an interrupt, then you are basically back to where you would have been under older versions of UNIX operating systems. Just be sure to use the *iod_ctlr* field of the *iocc_device* structures to address the device; compiling in funny constants will make your driver less portable.

6.4. Adding Non-Standard System Facilities

This section describes the work needed to augment *config*'s data base files for non-standard system facilities.

For *config*, non-standard facilities fall into two categories, those for kernel-profiling and those that are configuration-dependent. Files used for kernel profiling appear in the "files" files with a *profiling-routine* keyword. For example, the current profiling subroutines are found in a separate file with the following entry:

sys/subr_mcount.c **optional profiling-routine**

The *profiling-routine* keyword prohibits *config* from compiling the source file with the **−pg** option.

The keyword for the second category is *config-dependent*. This makes *config* compile the appropriate module with the global configuration parameters. This allows certain modules such as *machdep.c* to size system data structures based on the maximum users configured for the system.

APPENDIX A. CONFIGURATION FILE GRAMMAR

The following grammar is a compressed form of the actual *yacc*(1) grammar used by *config* to parse configuration files. Terminal symbols are shown all in upper case, literals are emboldened; optional clauses are enclosed in brackets "[" and "]"; zero or more instantiations are denoted with "*".

Configuration ::= [Spec ;]*

Spec ::= Config_spec
  | Device_spec
  | **trace**
  | /* lambda */

/* configuration specifications */

Config_spec ::= **machine** ID
  | **cpu** ID
  | **options** Opt_list
  | **ident** ID
  | System_spec
  | **timezone** [−] NUMBER [**dst** [NUMBER]]
  | **timezone** [−] FPNUMBER [**dst** [NUMBER]]
  | **maxusers** NUMBER

/* system configuration specifications */

System_spec ::= **config** ID System_parameter [System_parameter]*

System_parameter ::= swap_spec | root_spec | dump_spec | arg_spec

swap_spec ::= **swap** [**on**] swap_dev [**and** swap_dev]*

swap_dev ::= dev_spec [**size** NUMBER]

root_spec ::= **root** [**on**] dev_spec

dump_spec ::= **dumps** [**on**] dev_spec

arg_spec ::= **args** [**on**] dev_spec

dev_spec ::= dev_name | major_minor

major_minor ::= **major** NUMBER **minor** NUMBER

dev_name ::= ID [NUMBER [ID]]

/* option specifications */

Opt_list ::= Option [, Option]*

Option ::= ID [= Opt_value]

Opt_value ::= ID | NUMBER

```
/* device specifications */

Device_spec ::= device Dev_name Dev_info Int_spec
        | master Dev_name Dev_info
        | disk Dev_name Dev_info
        | tape Dev_name Dev_info
        | controller Dev_name Dev_info [ Int_spec ]
        | pseudo-device Dev [ NUMBER ]

Dev_name ::=  Dev NUMBER

Dev ::=  uba | mba | ID

Dev_info ::=  Con_info [ Info ]*

Con_info ::=  at Dev NUMBER
        | at nexus NUMBER

Info ::=  csr NUMBER
        | drive NUMBER
        | slave NUMBER
        | flags NUMBER

Int_spec ::=  vector ID [ ID ]*
        | priority NUMBER
```

Lexical Conventions

The terminal symbols are loosely defined as:

ID
: One or more alphabetics, either upper or lower case, and underscore.

NUMBER
: Similar to the C language specification for an integer number. That is, a leading "0x" indicates a hexadecimal value, a leading "0" indicates an octal value, otherwise the number is expected to be a decimal value. Hexadecimal numbers may use either upper or lower case alphabetics.

FPNUMBER
: A floating point number without exponent. That is a number of the form "nnn.ddd", where the fractional component is optional.

In special instances a question mark (?), can be substituted for a "NUMBER" token. This is used for wildcarding in device interconnection specifications.

Comments in configuration files being with a "#" character; the remainder of the line is discarded.

A specification is interpreted as a continuation of the previous line if the first character of the line is tab.

APPENDIX B. RULES FOR DEFAULTING SYSTEM DEVICES

When *config* processes a "config" rule that does not fully specify the location of the root file system, paging area(s), device for system dumps, and device for argument list processing it applies a set of rules to define those values left unspecified. The following list of rules are used in defaulting system devices.

(1) If a root device is not specified, the swap specification must indicate a "generic" system is to be built.

(2) If the root device does not specify a unit number, it defaults to unit 0.

(3) If the root device does not include a partition specification, it defaults to the "a" partition.

(4) If no swap area is specified, it defaults to the "b" partition of the root device.

(5) If no device is specified for processing argument lists, the first swap partition is selected.

(6) If no device is chosen for system dumps, the first swap partition is selected (see below to find out where dumps are placed within the partition).

The following table summarizes the default partitions selected when a device specification is incomplete (e.g. "hd0").

Type	Partition
root	"a"
swap	"b"
args	"b"
dumps	"b"

Multiple swap/paging areas

When multiple swap partitions are specified, the system treats the first specified as a "primary" swap area that is always used. The remaining partitions are then interleaved into the paging system at the time a *swapon*(2) system call is made. This is normally done at boot time with a call to *swapon*(8) from the /etc/rc file.

System dumps

System dumps are automatically taken after a system crash, provided the device driver for the "dumps" device supports this. The dump contains the contents of memory, but not the swap areas. Normally the dump device is a disk; the information is copied to a location near the back of the partition. The dump is placed in the back of the partition because the primary swap and dump device are commonly the same device and this allows the system to be rebooted without immediately overwriting the saved information. When a dump has occurred, the system variable *dumpsize* is set to a non-zero value indicating the size (in bytes) of the dump. The *savecore*(8) program then copies the information from the dump partition to a file in a "crash" directory and also makes a copy of the system that was running at the time of the crash (usually "/vmunix"). The offset to the system dump is defined in the system variable *dumplo* (a sector offset from the front of the dump partition). The *savecore* program operates by reading the contents of *dumplo*, *dumpdev*, and *dumpmagic* from /dev/kmem, then comparing the value of *dumpmagic* read from /dev/kmem to that located in corresponding location in the dump area of the dump partition. If a match is found, *savecore* assumes a crash occurred and reads *dumpsize* from the dump area of the dump partition. This value is then used in copying the system dump.

Refer to *savecore*(8) for more information about its operation.

The value *dumplo* is calculated to be

dumpdev-size − MAXDUMP

where *dumpdev-size* is the size of the disk partition where system dumps are to be placed, and MAXDUMP is 4 megabytes. If the disk partition is not large enough to hold a 4-megabyte dump, *dumplo* is set to 0 (the front of the partition).

APPENDIX C. SAMPLE CONFIGURATION FILE

The following sample configuration file is developed in Chapter 5; it is included here for completeness.

```
#
# Sample Configuration File for the IBM RT PC
#
machine         ca
cpu             "IBMRTPC"
ident           SAMPLE
timezone        8 dst
maxusers        16
options         INET

config          vmunix          root on hd0 swap on hd0 and hd1

controller      iocc0           at nexus ?
device          un0             at iocc0 csr 0xf4080000  priority 6
controller      hdc0            at iocc0 csr 0xf00001f0 priority 14
controller      hdc1            at iocc0 csr 0xf0000170 priority 14
controller      fdc0            at iocc0 csr 0xf00003f2 priority 6
controller      stc0            at iocc0 csr 0xf00001e8 priority 12
disk            hd0             at hdc0 drive 0
disk            hd1             at hdc0 drive 1
disk            hd2             at hdc1 drive 0
device          asy0            at iocc0 csr 0xf0001230 priority 9
device          asy1            at iocc0 csr 0xf0002230  priority 10
device          asy4            at iocc0 csr 0xf00003f8  priority 4
device          lp0             at iocc0 csr 0xf00003bc  priority 7
device          fd0             at fdc0 drive 0
tape            st0             at stc0 drive 0
pseudo-device   mono
pseudo-device   pty
pseudo-device   loop
pseudo-device   inet
pseudo-device   ether
pseudo-device   ms
pseudo-device   aed
pseudo-device   apaeight
pseudo-device   apaeightc
pseudo-device   apasixteen
```

APPENDIX D. KERNEL DATA STRUCTURE SIZING RULES

Certain system data structures are sized at compile time according to the maximum simultaneous users expected, while others are calculated at boot time based on the physical resources present. This appendix lists both sets of rules and also includes some hints on changing built-in limitations on certain data structures.

Compile time rules

The file /sys/conf/param.c contains the definitions of almost all data structures sized at compile time. This file is copied into the directory of each configured system to allow configuration-dependent rules and values to be maintained. The rules implied by its contents are summarized below (here MAXUSERS refers to the value defined in the configuration file in the "maxusers" rule).

nproc

The maximum number of processes which may be running at any time. It is defined to be 20 + 8 * MAXUSERS and referred to in other calculations as NPROC.

ntext

The maximum number of active shared text segments. Defined as 24 + MAXUSERS + NETSLOP, where NETSLOP is 20 when the Internet protocols are configured in the system and 0 otherwise. The added size for supporting the network is to take into account the numerous server processes that are likely to exist.

ninode

The maximum number of files in the file system that may be active at any time. This includes files in use by users, as well as directory files being read or written by the system and files associated with bound sockets in the 4.2/RT ipc domain. This number is defined as (NPROC + 16 + MAXUSERS) + 32.

nfile

The number of "file table" structures. One file table structure is used for each open, unshared, file descriptor. Multiple file descriptors may reference a single file table entry when they are created through a *dup* call, or as the result of a *fork*. This value is defined to be

16 * (NPROC + 16 + MAXUSERS) / 10 + 32 + 2 * NETSLOP

where NETSLOP is defined as for **ntext**.

ncallout

The number of "callout" structures. One callout structure is used per internal system event handled with a timeout. Timeouts are used for terminal delays, watchdog routines in device drivers, protocol timeout processing, etc. This number is defined as 16 + NPROC.

nclist

The number of "c-list" structures. C-list structures are used in terminal I/O. This number is defined as 100 + 16 * MAXUSERS.

nmbclusters

The maximum number of pages that may be allocated by the network. This is defined as 256 (a quarter megabyte of memory) in /sys/h/mbuf.h. In practice, the network rarely uses this much memory. It starts off by allocating 64 kilobytes of memory, and then requests more as required. This value represents an upper bound.

nquota
The number of "quota" structures allocated. Quota structures are present only when disc quotas are configured in the system. One quota structure is kept per user. This number is defined to be (MAXUSERS * 9) / 7 + 3.

ndquot
The number of "dquot" structures allocated. Dquot structures are present only when disc quotas are configured in the system. One dquot structure is required per user, per active file system quota. That is, when a user manipulates a file on a file system on which quotas are enabled, the information regarding the user's quotas on that file system must be in-core. This information is cached, so that not all information must be present in-core all the time. Dquot is defined as (MAXUSERS * NMOUNT) / 4 + NPROC, where NMOUNT is the maximum number of mountable file systems.

Run-time calculations

The most important data structures sized at run-time are those used in the buffer cache. Allocation is done by swiping physical memory (and the associated virtual memory) immediately after the system has been started up; look in the file /sys/ca/machdep.c.

The buffer cache is comprised of several "buffer headers" and a pool of pages attached to these headers. Buffer headers are divided into two categories, those used for swapping and paging and those used for normal file I/O. The system tries to allocate 10% of available physical memory for the buffer cache (where *available* does not count that space occupied by the system's text and data segments). If the result is fewer than 16 pages of memory allocated, then 16 pages are allocated. This value is kept in the initialized variable *bufpages* so that it may be patched in the binary image (to allow tuning without recompiling the system). Adequate file I/O buffer headers are then allocated to allow each to hold 2 pages each, and half as many swap I/O buffer headers are then allocated. The number of swap I/O buffer headers is constrained to be no more than 256.

System size limitations

As distributed, the sum of the virtual sizes of the core-resident processes is limited to 512M bytes. The size of the text, and data segments of a single process are currently limited to 24M bytes each, and the stack segment size is limited to 512K bytes as a soft, user-changeable limit, and may be increased to 24M with the *setrlimit*(2) system call. If these are insufficient, they can be increased by changing the constants MAXTSIZ, MAXDSIZ and MAXSSIZ in the file /sys/ca/vmparam.h, as well as changing the definitions in /sys/h/dmap.h and /sys/h/text.h. In making this change, be sure you have adequate paging space. As normally configured, the system has only 16M bytes per paging area. The best way to get more space is to provide multiple, thereby interleaved, paging areas.

To increase the amount of resident virtual space possible, you can alter the constant USRPTSIZE (in /sys/ca/vmparam.h). To allow 1 gigabyte of resident virtual space one would change the 1 to a 2.

Because the file system block numbers are stored in page table *pg_blkno* entries, the maximum size of a file system is limited to 2^19 2048 byte blocks. Thus no file system can be larger than 1 gigabyte.

The count of mountable file systems is limited to 15. This should be sufficient. If you have many disks, it makes sense to make some of them single file systems; the paging areas don't count in this total. To increase this, it will be necessary to change the core-map /sys/h/cmap.h, since a 4 bit field is used here. The size of the core-map then expands to 16 bytes per 1024 byte page. (Remember to change MSWAPX and NMOUNT in /sys/h/param.h.)

The maximum value NOFILE (open files per process limit) can be raised to is 30 because of a bit field in the page table entry in /sys/machine/pte.h.

Assembler Reference Manual for 4.2/RT

ABSTRACT

This article is an updated version of an article entitled *Berkeley VAX/UNIX Assembler Reference Manual,* written in November 1979 by John F. Reiser and Robert R. Henry and revised in February 1983. The original article, which is in Volume 2C of the *UNIX Programmer's Manual,* has been rewritten and includes additions and changes for 4.2/RT and corrections where appropriate.

1. INTRODUCTION

This document describes the usage and input syntax of the 4.2/RT assembler for the IBM RT PC, *as*. *As* assembles the code produced by the C compiler. This article is intended for those writing a compiler or maintaining the assembler; it is not a user's guide for writing assembler code.

Examples of syntax in this article use the following conventions:

- [Argument] means that the specified argument is optional; 0 or more instances may be included.
- Words in **boldface** must appear literally.
- Words in *italics* represent specific values to be supplied.

2. USAGE

As is invoked with these command arguments:

as [−**LVWRDT**] [−**t** *directory*] [−**o** *outfile*] [*name*$_1$] . . . [*name*$_n$]

The arguments are explained below:

-L Instructs the assembler to save labels beginning with an "L" in the symbol table portion of the file specified as *outfile.* Labels are not saved by default, as the default action of the link editor *ld* is to discard them anyway.

-V Tells the assembler to place its interpass temporary file in virtual memory. In normal circumstances, the system manager will decide where the temporary file should lie. Experiments with a temporary file of 115 kbytes have shown this option to have a negligible (1-2%) effect on assembly time on an unloaded machine.

-W Turns off all warning error reporting.

-R Make initialized data segments read-only by concatenating them to the text segments. This obviates the need to run editor scripts on assembler source to "read−only" fix initialized data segments. Uninitialized data (via **.lcomm** and **.comm** directives) are still assembled into the bss segment.

-D Prints assembler debugging information and dumps the symbol table, provided the assembler has been compiled with DEBUG defined.

-T Prints the token file, provided the assembler has been compiled with DEBUG defined. This information is useful when debugging the assembler.

-t Causes the assembler to place its single temporary file in *directory* instead of in /tmp , provided the −**V** flag is not set.

-o Causes the output to be placed in the file *outfile.* By default, the output of the assembler is placed in the file a.out in the current directory.

name$_{1\text{-}n}$ Causes input to be taken sequentially from the files *name*$_1$ . . . *name*$_n$. The files are not assembled separately; *name*$_1$ is effectively concatenated to *name*$_2$ so multiple definitions cannot occur among the input sources. By default, input is taken from the standard input.

Note: Arguments **-J** and **-d** are ignored.

3. LEXICAL CONVENTIONS

Assembler tokens include identifiers (alternatively, "symbols" or "names"), constants, and operators.

3.1. Identifiers

An identifier consists of a sequence of alphanumeric characters, including the special characters period (.), underscore (_), and dollar ($). The first character may not be a digit or a dollar sign. For all practical purposes, the length of identifiers is arbitrary; all characters are significant. All keywords, operation mnemonics, register names, and macro names are reserved and are not available as user-defined names.

3.2. Constants

3.2.1. Integral Constants

All integral (non floating point) constants are (potentially) 64 bits wide. Integral constants are initially evaluated to a full 64 bits, but are pared down by discarding high order copies of the sign bit and categorizing the number as a long (32 bits) or double-long (64 bits) integer. Numbers with less precision than 32 bits are treated as 32-bit quantities. *As* cannot perform arithmetic on constants larger than 32 bits and supports 64-bit integers only so they can be used to fill initialized data space.

The digits are "0123456789abcdefABCDEF" with the obvious values.

A decimal constant consists of a sequence of digits without a leading zero.

An octal constant consists of a sequence of digits with a leading zero.

A hexadecimal constant consists of the characters "0x" (or "0X") followed by a sequence of digits.

A single-character constant consists of a single quote (') followed by an ASCII character, including ASCII newline. The constant's value is the code for the given character.

3.2.2. Floating Point Constants

IEEE single and double precision constants are supported by the **.float** and **.double** directives respectively. The *atof*(3) man page describes the range of representable values and their syntax. There is presently no support for IEEE double extended precision constants. For a description of the IEEE representations, please see the *IEEE Standard 754 for Binary Floating Point Arithmetic.* The assembler uses the library routine *atof*(3) to convert floating point numbers.

The operand field syntax of .float and .double is:

$0[expe]([+-])\ [dec]^{+}(.)([dec]^{*})([expt]([+-])([dec]^{+}))$

where:

expe	An exponent delimiter and type specification character (fFdD).
dec	A decimal digit (0 1 2 3 4 5 6 7 8 9).
expt	A type specification character (eEfFdD).
x^{*}	0 or more occurrences of *x*.
x^{+}	1 or more occurrences of *x*.

The standard semantic interpretation is used for the signed integer, fraction and signed power of 10 exponent. If the exponent delimiter is specified, it must be either an "e" or "E", or must agree with the initial type specification character that is used. A .double constant must have d or D specified as its type specification character; a .float constant must have f or F specified as its type specification character.

Collectively, all floating point numbers, together with double-long integral numbers, are called "bignums". When *as* requires a bignum, a 32-bit scalar quantity may also be used.

3.2.3. String Constants

A string constant is defined using the same syntax and semantics as the C language uses. Strings begin and end with a double quote ("). All C backslash conventions are observed. Strings are known by their value and their length; the assembler does not implicitly end strings with a null byte.

3.3. Operators

There are several single-character operators; see Section 6.1.

3.4. Blanks

Blank and tab characters may be interspersed freely between tokens, but may not be used within tokens (except character constants). A blank or tab is required to separate adjacent identifiers or constants not otherwise separated.

3.5. Single Line Comments

The character "#" introduces a comment which extends through the end of the line. Comments starting in column 1, having the format *"# expression string"*, are interpreted as an indication that the assembler is now assembling file *string* at line *expression*. Thus, one can use the C preprocessor on an assembly language source file, and use the *#include* and *#define* preprocessor directives. Other comments may not start in column 1 if the assembler source is given to the C preprocessor because the preprocessor will misinterpret them. Comments are otherwise ignored by the assembler.

To retain compatibility with existing .s files, comments beginning with "|" are also accepted. However, this use is deprecated, and support for this feature will be removed in subsequent releases.

3.6. C Style Comments

The assembler will recognize C style comments, introduced with the prologue /* and ending with the epilogue */. C style comments may extend across multiple lines and are the preferred comment style to use if you choose to use the C preprocessor.

If a C style comment does extend across "n" lines, the line numbers in any subsequent error messages generated by the assembler will be low by n-1 lines, since the assembler increments the line count only once for a multiple C style comment.

4. SEGMENTS AND LOCATION COUNTERS

Assembled code and data fall into three segments: the text segment, the data segment, and the bss segment. The operating system makes some assumptions about the content of these segments; the assembler does not. Within the text and data segments there are a number of sub-segments, distinguished by number ("**text** 0", "**text** 1", "**data** 0", "**data** 1" , . . .). Currently there are four subsegments each in text and data. The subsegments are for programming convenience only.

Before writing the output file, the assembler zero-pads each text subsegment to a multiple of eight bytes and then concatenates the subsegments in order to form the text segment; an analogous operation is done for the data segment. Requesting that the loader define symbols and storage regions is the only action allowed by the assembler with respect to the bss segment. Assembly begins in "**text** 0".

Associated with each (sub)segment is an implicit location counter which begins at zero and is incremented by 1 for each byte assembled into the (sub)segment. There is no way to explicitly reference a location counter. Note that the location counters of subsegments other than "**text** 0" and "**data** 0" behave peculiarly due to the concatenation used to form the text and data segments.

5. STATEMENTS

A source program is composed of a sequence of statements. Statements are separated by newlines or by semicolons. There are two kinds of statements: null statements and keyword statements. Either kind of statement may be preceded by one or more labels.

5.1. Named Labels

A named label consists of a name followed by a colon. The effect of a named label is to assign the current value and type of the location counter to the name. An error is indicated in pass 1 if the name is already defined; an error is indicated in pass 2 if the value assigned changes the definition of the label.

Named labels beginning with an "L" are not retained in the a.out symbol table unless the **−L** option is in effect.

5.2. Numeric Local Labels

A numeric label consists of a digit between 0 and 9 followed by a colon. A numeric label defines temporary symbols of the form "*n*b" and "*n*f" where *n* is the digit of the label. As in the case of named labels, a numeric label assigns the current value and type of the location counter to the temporary symbol. However, several numeric labels with the same digit may be used within the same assembly. References to symbols of the form "*n*b" refer to the first numeric label *n*: *b*ackward from the reference; "*n*f" symbols refer to the first numeric label *n*: *f*orward from the reference.

As turns local labels into labels of the form L*n*\001m for internal purposes.

5.3. Null Statements

A null statement is an empty statement ignored by the assembler. A null statement may be labeled, however.

5.4. Keyword Statements

A keyword statement begins with one of the many predefined keywords known to *as*; the syntax of the remainder of the statement depends on the keyword. All instruction opcodes, listed in Section 8, are keywords. The remaining keywords are assembler pseudo-operations, also called "directives." The pseudo-operations are listed in Section 7, together with the syntax they require.

6. EXPRESSIONS

An expression is a sequence of symbols representing a value. Its constituents are identifiers, constants, operators, and parentheses. Each expression has a type.

All operators in expressions are fundamentally binary in nature. Arithmetic is two's complement and has 32 bits of precision. *As* cannot perform arithmetic operations on floating point numbers or on double-long integral numbers. There are four levels of precedence, listed here from lowest precedence level to highest:

precedence	operators
binary	+ −
binary	& ^ !
binary	* / %
unary	− ~

All operators of the same precedence are evaluated strictly left to right, except for the evaluation order enforced by parentheses.

6.1. Expression Operators

The operators are:

operator	meaning
+	addition
−	(binary) subtraction
*	multiplication
/	division
%	modulo
−	(unary) two's complement
&	bitwise and
^	bitwise exclusive or
!	bitwise or not
~	bitwise ones' complement
>	logical right shift
> >	logical right shift
<	logical left shift
< <	logical left shift

Expressions may be grouped with parentheses.

6.2. Data Types

Every user-defined symbol has one of the following types. The type propagation rules in the next section describe how expression types are derived from symbol types.

undefined Upon first encounter, each symbol is undefined unless its first encounter defines it. It may become undefined if it is assigned an undefined expression. The assembler changes all undefined types to undefined external just prior to pass 2.

undefined external
A symbol which is declared **.globl** but not defined in the current assembly is an undefined external. If such a symbol is declared, the link editor *ld* must be used to load the assembler's output with another routine that defines the undefined reference.

absolute An absolute symbol is defined in a **.set** by an expression of type absolute. Constants have type absolute.

text A symbol appearing as a label in a text segment has type text, as does a symbol defined in a **.set** by an expression of type text. The value of a text symbol is measured with respect to the beginning of the text segment of the program. If the assembler output is link-edited, its text symbols may change in value since the program need not be the first in the link editor's output.

data A symbol appearing as a label in a data segment has type data, as does a symbol defined in a **.set** by an expression of type data. The value of a data symbol is measured with respect to the origin of the data segment of a program. The value of a data symbol may change during a subsequent link-editor run since previously loaded programs may have data segments.

bss A symbol defined in a **.comm** or **.lcomm** directive has type bss, as does a symbol defined in a **.set** by an expression of type bss. The value of a bss symbol is measured from the beginning of the bss segment of a program. The value of a bss symbol may change during a subsequent link-editor run, since previously loaded programs may have bss segments.

external absolute, text, data, or bss
Symbols declared **.globl** and defined within an assembly as absolute, text, data, or bss types may be used exactly as if they were not declared **.globl**; however, their value and type are available to the link editor so that the program may be loaded with others that reference these symbols.

6.3. Type Propagation in Expressions

When operands are combined by expression operators, the result has a type which depends on the types of the operands and on the operator. The rules involved are complex to state but were intended to be sensible and predictable. For purposes of expression evaluation, the important types are:

undefined
absolute
text
data
bss
undefined external
relocatable: any of text, data, bss, or undefined external

The combination rules are:

(1) If one of the operands is undefined, the result is undefined.

(2) If both operands are absolute, the result is absolute.

(3) An absolute operand may be added to or subtracted from any other type, and the type of the result is that of the other operand.

(4) An operand of type text, data, or bss may be subtracted from an operand having the same type, and the type of the result is absolute.

(5) Any other combination is an error.

7. PSEUDO-OPERATIONS (DIRECTIVES)

The keywords listed below introduce pseudo-operations (directives) to influence the later behavior of the assembler, define symbols, or create data. They are grouped below into functional categories.

7.1. Interface to a Previous Pass

.ABORT

As soon as the assembler sees this directive, it ignores all further input (but it does read to the end of file) and aborts the assembly. No files are created. It is anticipated that this would be used in a pipe interconnected version of a compiler, where the first major syntax error would cause the compiler to issue this directive, saving unnecessary work in assembling code that would have to be discarded anyway.

.file *string*

This directive causes the assembler to think it is in file *string*, so that error messages reflect the proper source file.

.line *expression*

This directive causes the assembler to think it is on line *expression* so that error messages reflect the proper source line.

The only effect of assembling multiple files specified in the command string is to insert the *file* and *line* directives, with the appropriate values, at the beginning of the source from each file.

# *expression string*

This is the only instance where a comment is meaningful to the assembler. The "#" must be in the first column. This meta comment causes the assembler to believe it is on line *expression.* The second argument, if included, causes the assembler to believe it is in file *string*; otherwise the current file name does not change.

7.2. Location Counter Control

.data [*expression*]
.text [*expression*]

These two directives cause the assembler to begin assembling into the indicated text or data subsegment. If specified, *expression* must be defined and absolute; an omitted expression is treated as zero. Assembly starts in the **.text** 0 subsegment.

The directives **.align** and **.org** also control the placement of the location counter.

While the comments within the assembler may refer to the location counter as "." or "dot", there is no explicit reference allowed to the location counter. Numeric local labels may be used with almost equal convenience and more predictable results.

7.3. Filled Data

.align *align_expr*

The location counter is adjusted so that the *align_expr* lowest bits of the location counter become zero. This is done by assembling from 0 to 2^{align_expr} -1 bytes of 0. Thus ".align 2" pads by null bytes to make the location counter evenly divisible by 4. The *align_expr* must be defined, absolute, nonnegative, and less than 16.

Warning: the subsegment concatenation convention and the current loader conventions may not preserve attempts at aligning to more than 3 low-order zero bits.

.org *org_expr*[,*fill_expr*]

The location counter is set equal to the value of *org_expr*, which must be of type text or data and greater than the current value of that segment's location counter. Space between the current value of the location counter and the desired value are filled with bytes taken from the low order byte of *fill_expr*, which must be absolute and defaults to 0.

.space *space_expr*[,*fill_expr*]

The location counter is advanced by *space_expr* bytes. *Space_expr* must be defined and absolute. The space is filled in with bytes taken from the low order byte of *fill_expr*, which must be defined and absolute. *Fill_expr* defaults to 0. The **.fill** directive is a more general way to accomplish the **.space** directive.

.fill *rep_expr, size_expr, fill_expr*

All three expressions must be absolute. *Fill_expr*, treated as an expression of size *size_expr* bytes, is assembled and replicated *rep_expr* times. The effect is to advance the current location counter *rep_expr* * *size_expr* bytes. *Size_expr* must be between 1 and 8.

7.4. Initialized Data

.byte	*expr*[,*expr*]. . .
.short	*expr*[,*expr*]. . .
.int	*expr*[,*expr*]. . .
.long	*expr*[,*expr*]. . .

Expr represents an expression. Expressions are truncated to the size indicated by the keyword in the table below, and assembled in successive locations. Non-absolute expressions in a **.byte** or **.short** engender a warning message.

keyword	length (bits)
.byte	8
.short	16
.int	32
.long	32

Each expression may optionally be of the form:

$expression_1 : expression_2$

In this case, the value of *expression*$_2$ is truncated to *expression*$_1$ bits, and assembled in the next *expression*$_1$ bit field which fits in the natural data size being assembled. Bits which are skipped because a field does not fit are filled with zeros. Thus, "**.byte** 123" is equivalent to "**.byte** 8:123", and "**.byte** 3:1,2:1,5:1" assembles two bytes, containing the values 0x28 and 0x08.

.dlong *number*[,*number*]. . .
.float *number*[,*number*]. . .
.double *number*[,*number*]. . .

These initialize bignums (see Section 3.2.2) in successive locations whose size is a function of the keyword. The type of the bignum (determined by the exponent field, or lack thereof) may not agree with the type implied by the keyword. The following table shows the keywords, their size, and the data types for the bignums they expect.

keyword	format	length (bits)	valid *number* (s)
.dlong	integral	64	integral
.float	ieee single	32	floating and integral
.double	ieee double	64	floating and integral

.ascii *string*[, *string*]. . .
.asciz *string*[, *string*]. . .

Each *string* in the list is assembled into successive locations, with the first letter in the string being placed into the first location, etc. The **.ascii** directive will not null terminate the string; the **.asciz** directive will null terminate the string. (Recall that strings are known by their length and need not be terminated with a null, and that the C conventions for escaping are understood.) The **.ascii** directive is identical to:

  .byte *string*$_0$, *string*$_1$, . . .

.comm *name, expression*

Provided the *name* is not defined elsewhere, its type is made "undefined external", and its value is *expression*. In fact the *name* behaves in the current assembly just like an undefined external. However, the link editor *ld* has been special-cased so that all undefined external symbols that have a non-zero value are defined to lie in the bss segment, and space is reserved after the symbol to hold *expression* bytes.

.lcomm *name, expression*

Expression bytes will be allocated in the bss segment and *name* assigned the location of the first byte, but the *name* is not declared as global and hence will be unknown to the link editor.

.globl *name*

This directive makes *name* external. If it is otherwise defined (by **.set** or by appearance as a label) it acts within the assembly exactly as if the **.globl** directive were not given; however, the link editor may be used to combine this object module with other modules referring to this symbol.

Conversely, if the given symbol is not defined within the current assembly, the link editor can combine the output of this assembly with that of others which define the symbol.

.set *name, expression*

The (*name, expression*) pair is entered into the symbol table. Multiple **.set** statements with the same name are legal; the most recent value replaces all previous values.

.lsym *name, expression*

A unique instance of the (*name, expression*) pair is created in the symbol table. This mechanism can be used to pass local symbol definitions to the link editor and debugger. Note that *name* may not be referenced.

.stabs *string*, $expr_1$, $expr_2$, $expr_3$, $expr_4$
.stabn $expr_1$, $expr_2$, $expr_3$, exr_4
.stabd $expr_1$, $expr_2$, $expr_3$

The **.stab***x* directives place symbols in the symbol table for the symbolic debugger, *dbx*. A "stab" is a *s*ymbol *tab*le entry. The **.stabs** is a string stab, the **.stabn** is a stab not having a string, and the **.stabd** is a "dot" stab that implicitly references "dot", the current location counter.

The *string* in the **.stabs** directive is the name of a symbol. If the symbol name is zero, the **.stabn** directive may be used instead.

The other expressions are stored in the name list structure of the symbol table and preserved by the loader for reference by *dbx*; the values of the expressions are peculiar to formats required by *dbx*.

expr$_1$ Is used as a symbol table tag (nlist field *n_type*).

expr$_2$ Is always zero (nlist field *n_other*).

expr$_3$ Is used for either the source line number, or for a nesting level (nlist field *n_desc*).

expr$_4$ Is used as tag specific information (nlist field *n_value*). In the case of the **.stabd** directive, this expression is nonexistent, and is taken to be the value of the location counter at the following instruction. Since there is no associated name for a **.stabd** directive, it can be used only in circumstances where the name is zero. The effect of a **.stabd** directive can be achieved by one of the other **.stab**x directives in the following manner:

> **.stabn** $expr_1$, $expr_2$, $expr_3$, LL_n
> LL*n*:

The **.stabd** directive is preferred, because it does not clog the symbol table with labels used only for the stab symbol entries.

7.5. Addressability

.using *expr,register,. . .*

The **.using** directive tells the assembler that it can rely on the value in a register for the purpose of creating base + displacement addresses for machine instructions.

Expr may be any relocatable expression of type text or data. The register is assumed to contain an address pointing to the storage location described by the relocatable expression. Each additional specified register is assumed to contain an address 0x8000 bytes greater than the previous register.

There may be one **.using** specified for each text subsegment and one for each data subsegment (i.e. up to eight **.using**'s may be in effect at any time). If a **.using** is not provided for a .text or for a .data subsegment but is provided for a lower-numbered text or data subsegment, the one for the lower-numbered subsegment will be used. If no **.using** is provided for any text subsegment, reference to an address of type text encodes a warning message and register 11 is assumed to point to the beginning of the text 0 subsegment. If no **.using** is provided for any data subsegment, reference to an address of type data engenders an error message. If a proper register and displacement cannot be formed from a **.using** statement, an error message is issued.

If a second **.using** is specified while one is active within the same subsegment, the second replaces the first. A **.using** followed by a relocatable expression without a register unassigns the base register.

Symbols in the relocatable expression need not be defined before the appearance of the **.using** directive.

7.6. Literal Operands

The following construct may be used in machine instructions wherever a relocatable instruction operand may be used:

$*.data-directive expression*

The arguments are explained below:

data-directive	Any of **.byte**, **.short**, **.int**, **.long**, **.dlong**, **.float**, **.double**, **.ascii**, or **.asciz.**
expression	Any single expression that is legal for the respective assembler directive.

The following lines show examples of literals:

```
        lc      r1,$.byte 0x18
        lh      r2,$.short (4< <8)
        l       r2,$.int 123456
```

The line:

```
        l       r7,$.long root
```

is equivalent to:

```
        l       r7,Z00001
        . . .
Z00001:.long    root
```

Literals are accumulated into a pool and duplicates are removed. Literals are considered duplicates when they are written in exactly the same way; constants which assemble to the same value but which have different source forms are different literals, except that .long and .int are considered to be equal. String literals are never considered to be equal. The literal pool is sorted such that the items with the more restrictive alignment are placed first. The beginning of the literal pool is aligned to the boundary implied by the first literal in the pool.

.ltorg

This directive indicates the start of a literal pool and causes the accumulated literal values to be emitted. The **.ltorg** directive can appear in either a text or data segment, and it can appear more than once. If literals are used and no **.ltorg** follows, a warning will be issued and the literals will be emitted at the end of the .text 0 subsegment.

8. MACHINE INSTRUCTIONS

This section describes the machine instructions, extended branch mnemonics, and macro instructions supported by *as*.

8.1. Summary of Machine Instructions

The symbols used to describe the source syntax are:

abs	An absolute expression representing a displacement from a base.
f	An absolute value representing a register bit position.
i	An absolute expression representing an immediate value, optionally preceded by a "$".
lbl	A name of type text, data, or undefined external.
ra,rb,rc	Register expressions. A register expression is one of the predefined symbols r0, . . . r15, sp, or a "%" followed by an absolute in the range 0-15. sp is equivalent to r1.
reloc	An address operand of one of the following forms: l abs(register-expression) l $literal expn An expression of type text or data covered by a base register defined in a ".using" directive.

The following symbols are used to show the assembled result. A character repeated indicates that the field is wider that one hex digit.

a,b,c	Registers ra, rb, and rc.
f	A register bit position.
n	A numeric field.
d	A displacement from a register or the current location.

Most numeric fields and displacements represent sign-extended two's complement quantities. In the Operations column of the following table, "(unsigned)" indicates instructions that do not sign-extend.

Source Syntax		Assembled Format	Operation
a	ra,rb	e1ab	Add
abs	ra,rb	e0ab	Absolute
ae	ra,rb	f1ab	Add Extended
aei	ra,rb,i	d1ab nnnn	Add Extended Immediate
ai	ra, [rb,] i		(Macro) See Section 8.3
ail	ra,rb,i	c1ab nnnn	Add Immediate Long
ais	ra,i	90an	Add Immediate Short
bala	lbl	8ann nnnn	Branch and Link Absolute (unsigned)
balax	lbl	8bnn nnnn	Branch and Link Absolute with Execute (unsigned) **
bali	ra,lbl	8cad dddd	Branch and Link Immediate
balix	ra,lbl	8dad dddd	Branch and Link Immediate with Execute **
balr	ra,rb	ecab	Branch And Link Register
balrx	ra,rb	edab	Brand And Link Register with Execute **
bb	f,lbl	8efd dddd	Branch on Bit
bbr	f,ra	eefa	Branch on Bit
bbrx	f,ra	effa	Branch on Bit with Execute
bbx	f,lbl	8ffd dddd	Branch on Bit with Execute
bnb	f,lbl	88fd dddd	Branch on Not Bit
bnbr	f,ra	e8fa	Branch on Not Bit
bnbrx	f,ra	e9fa	Branch on Not Bit with Execute
bnbx	f,lbl	89fd dddd	Branch on Not Bit with Execute
c	ra,rb	b4ab	Compare
ca16	ra,rb	f3ab	Compute Address 16-bit
cal	ra,reloc	c8ab dddd	Compute Address Lower Half
cal16	ra,reloc	c2ab dddd	Compute Address Lower Half 16-bit (unsigned)
cas	ra,rb,rc	6abc	Compute Address Short
cau	ra,reloc	d8ab dddd	Compute Address Upper Half (unsigned)
ci	ra, i		(Macro) See Section 8.3
cil	ra,i	d40a nnnn	Compare Immediate Long
cis	ra,i	94an	Compare Immediate Short
cl	ra,rb	b3ab	Compare Logical
cli	ra, i		(Macro) See Section 8.3
clil	ra,i	d30a nnnn	Compare Logical Immediate Long
clrbl	ra,i	99an	Clear Bit Lower
clrbu	ra,i	98an	Clear Bit Upper
clrsb	ra,i	95an	Clear SCR Bit
clz	ra,rb	f5ab	Count Leading Zeros
d	ra,rb	b6ab	Divide Step
dec	ra,i	93an	Decrement
exts	ra,rb	b1ab	Extend Sign
get*	ra,$expr		(Macro) See Section 8.3
get*	ra,reloc		(Macro) See Section 8.3
inc	ra,i	91an	Increment
ior	ra,reloc	cbab dddd	Input/Output Read (unsigned)
iow	ra,reloc	dbab dddd	Input/Output Write (unsigned)
jb	f,lbl	08dd to 0fdd	Jump on Bit
jnb	f,lbl	00dd to 07dd	Jump on Not Bit

** If a two-byte instruction follows a Branch and Link with Execute, *as* appends a 'jnop'.

Source Syntax		Assembled Format	Operation
l	ra,reloc	cdab dddd	Load
lc	ra,reloc	ceab dddd	Load Character
lcs	ra,reloc	4dab	Load Character Short
lh	ra,reloc	daab dddd	Load Half
lha	ra,reloc	caab dddd	Load Half Algebraic
lhas	ra,reloc	5dab	Load Half Algebraic Short
lhs	ra,0(rb)	ebab	Load Half Short
lis	ra,i	a4an	Load Immediate Short
load*	ra,expr[(rb)]		(Macro) See Section 8.3
lm	ra,reloc	c9ab dddd	Load Multiple
lps	i,reloc	d0nb dddd	Load Program Status
ls	ra,reloc	7dab	Load Short
m	ra,rb	e6ab	Multiply Step
mc03	ra,rb	f9ab	Move Character 0 from 3
mc13	ra,rb	faab	Move Character 1 from 3
mc23	ra,rb	fbab	Move Character 2 from 3
mc30	ra,rb	fdab	Move Character 3 from 0
mc31	ra,rb	fcab	Move Character 3 from 1
mc32	ra,rb	ffab	Move Character 3 from 2
mc33	ra,rb	feab	Move Character 3 from 3
mfs	ra,rb	96ab	Move From SCR ra to register rb
mftb	ra,rb	bcab	Move From Test Bit
mftbil	ra,i	9dan	Move From Test Bit Immediate Lower
mftbiu	ra,i	9can	Move From Test Bit Immediate Upper
mr	ra,rb		(Macro) See Section 8.3
mts	ra,rb	b5ab	Move To SCR ra from register rb
mttb	ra,rb	bfab	Move To Test Bit
mttbil	ra,i	9fan	Move To Test Bit Immediate Lower
mttbiu	ra,i	9ean	Move To Test Bit Immediate Upper
n	ra,rb	e5ab	And
ni	ra,rb,i		(Macro) See Section 8.3
nilo	ra,rb,i	c6ab nnnn	And Immediate Lower Half Extended Ones (unsigned)
nilz	ra,rb,i	c5ab nnnn	And Immediate Lower Half Extended Zeros (unsigned)
niuo	ra,rb,i	d6ab nnnn	And Immediate Upper Half Extended Ones (unsigned)
niuz	ra,rb,i	d5ab nnnn	And Immediate Upper Half Extended Zeros (unsigned)
o	ra,rb	e3ab	Or
oi	ra,rb,i		(Macro) See Section 8.3
oil	ra,rb,i	c4ab nnnn	Or Immediate Lower Half (unsigned)
oiu	ra,rb,i	c3ab nnnn	Or Immediate Upper Half (unsigned)
onec	ra,rb	f4ab	Ones' Complement
put*	ra,reloc		(Macro) See Section 8.3
s	ra,rb	e2ab	Subtract
sar	ra,rb	b0ab	Shift Algebraic Right
sari	ra,i	a0an	Shift Algebraic Right Immediate
sari16	ra,i	a1an	Shift Algebraic Right Immediate plus 16
se	ra,rb	f2ab	Subtract Extended
setbl	ra,i	9ban	Set Bit Lower

Source Syntax		Assembled Format	Operation
setbu	ra,i	9aan	Set Bit Upper
setsb	ra,i	97an	Set SCR Bit
sf	ra,rb	b2ab	Subtract From
sfi	ra,rb,i	d2ab nnnn	Subtract From Immediate
shl	ra,i		(Macro) See Section 8.3
shla	ra,i		(Macro) See Section 8.3
shr	ra,i		(Macro) See Section 8.3
shra	ra,i		(Macro) See Section 8.3
si	ra,[rb,]i		(Macro) See Section 8.3
sil	ra,rb,i		(Macro) See Section 8.3
sis	ra,i	92an	Subtract Immediate Short
sl	ra,rb	baab	Shift Left
sli	ra,i	aaan	Shift Left Immediate
sli16	ra,i	aban	Shift Left Immediate plus 16
slp	ra,rb	bbab	Shift Left Paired
slpi	ra,i	aean	Shift Left Paired Immediate
slpi16	ra,i	afan	Shift Left Paired Immediate plus 16
sr	ra,rb	b8ab	Shift Right
sri	ra,i	a8an	Shift Right Immediate
sri16	ra,i	a9an	Shift Right Immediate plus 16
srp	ra,rb	b9ab	Shift Right Paired
srpi	ra,i	acan	Shift Right Paired Immediate
srpi16	ra,i	adan	Shift Right Paired Immediate plus 16
st	ra,reloc	ddab dddd	Store
stc	ra,reloc	deab dddd	Store Character
stcs	ra,reloc	1dab	Store Character Short
sth	ra,reloc	dcab dddd	Store Half
sths	ra,reloc	2dab	Store Half Short
stm	ra,reloc	d9ab dddd	Store Multiple
store*	ra,expr[(rb)],rc		(Macro) See Section 8.3
sts	ra,reloc	3dab	Store Short
svc	abs(ra)	c00a nnnn	Supervisor Call (unsigned)
tgte	ra,rb	bdab	Trap if Register Greater Than or Equal
ti	f,ra,i	ccfa nnnn	Trap on Condition Immediate
tlt	ra,rb	beab	Trap if Register Less Than
tsh	ra,reloc	cfab dddd	Test and Set Half
twoc	ra,rb	e4ab	Two's Complement
wait		f000	Wait
x	ra,rb	e7ab	Exclusive Or
xi	ra,rb,i		(Macro) See Section 8.3
xil	ra,rb,i	c7ab nnnn	Exclusive Or Immediate Lower Half (unsigned)
xiu	ra,rb,i	d7ab nnnn	Exclusive Or Immediate Upper Half (unsigned)

8.2. Extended Mnemonics: Branch on Bit

Source Syntax		Assembled Format	Operation
b	lbl	888d dddd	Branch
bc0	lbl	8ecd dddd	Branch on Carry 0
be	lbl	8ead dddd	Branch on Equal
beq	lbl	8ead dddd	Branch on Equal
bh	lbl	8ebd dddd	Branch on High
bhe	lbl	889d dddd	Branch on High or Equal
bl	lbl	8e9d dddd	Branch on Low
ble	lbl	88bd dddd	Branch on Low or Equal
bm	lbl	8e9d dddd	Branch on Minus
bnc0	lbl	88cd dddd	Branch on Not Carry 0
bne	lbl	88ad dddd	Branch on Not Equal
bnh	lbl	88bd dddd	Branch on Not High
bnl	lbl	889d dddd	Branch on Not Low
bnm	lbl	889d dddd	Branch on Not Minus
bno	lbl	88ed dddd	Branch on Not Overflow
bnp	lbl	88bd dddd	Branch on Not Plus
bntb	lbl	88fd dddd	Branch on Not Test Bit
bnz	lbl	88ad dddd	Branch on Not Zero
bo	lbl	8eed dddd	Branch on Overflow
bp	lbl	8ebd dddd	Branch on Plus
btb	lbl	8efd dddd	Branch on Test Bit
bz	lbl	8ead dddd	Branch on Zero
nop	lbl	8eod dddd	No Operation
bc0x	lbl	8fcd dddd	Branch on Carry 0 with Execute
beqx	lbl	8fad dddd	Branch on Equal with Execute
bex	lbl	8fad dddd	Branch on Equal with Execute
bhex	lbl	899d dddd	Branch on High or Equal with Execute
bhx	lbl	8fbd dddd	Branch on High with Execute
blex	lbl	89bd dddd	Branch on Low or Equal with Execute
blx	lbl	8f9d dddd	Branch on Low with Execute
bmx	lbl	8f9d dddd	Branch on Minus with Execute
bnc0x	lbl	89cd dddd	Branch on Not Carry 0 with Execute
bnex	lbl	89ad dddd	Branch on Not Equal with Execute
bnhx	lbl	89bd dddd	Branch on Not High with Execute
bnlx	lbl	899d dddd	Branch on Not Low with Execute
bnmx	lbl	899d dddd	Branch on Not Minus with Execute
bnox	lbl	89ed dddd	Branch on Not Overflow with Execute
bnpx	lbl	89bd dddd	Branch on Not Plus with Execute
bntbx	lbl	89fd dddd	Branch on Not Test Bit with Execute
bnzx	lbl	89ad dddd	Branch on Not Zero with Execute
box	lbl	8fed dddd	Branch on Overflow with Execute
bpx	lbl	8fbd dddd	Branch on Plus with Execute
btbx	lbl	8ffd dddd	Branch on Test Bit with Execute
bx	lbl	898d dddd	Branch with Execute
bzx	lbl	8fad dddd	Branch on Zero with Execute
nopx	lbl	8f8d dddd	No Operation with Execute

8.3. Extended Mnemonics: Branch Register

Source Syntax		Assembled Format	Operation
bc0r	ra	eeca	Branch on Carry 0
beqr	ra	eeaa	Branch on Equal
ber	ra	eeaa	Branch on Equal
bher	ra	e89a	Branch on High or Equal
bhr	ra	eeba	Branch on High
bler	ra	e8ba	Branch on Low or Equal
blr	ra	ee9a	Branch on Low
bmr	ra	ee9a	Branch on Minus
bnc0r	ra	e8ca	Branch on Not Carry 0
bner	ra	e8aa	Branch on Not Equal
bnhr	ra	e8ba	Branch on Not High
bnlr	ra	e89a	Branch on Not low
bnmr	ra	e89a	Branch on Not Minus
bnor	ra	e8ea	Branch on Not Overflow
bnpr	ra	e8ba	Branch on Not Plus
bntbr	ra	e8fa	Branch on Not Test Bit
bnzr	ra	e8aa	Branch on Not Zero
bor	ra	eeea	Branch on Overflow
bpr	ra	eeba	Branch on Plus
br	ra	e88a	Branch
btbr	ra	eefa	Branch on Test Bit
bzr	ra	eeaa	Branch on Zero
nopr	ra	ee8a	No Operation
bc0rx	ra	efca	Branch on Carry 0 with Execute
beqrx	ra	efaa	Branch on Equal with Execute
berx	ra	efaa	Branch on Equal with Execute
bherx	ra	e99a	Branch on High or Equal with Execute
bhrx	ra	efba	Branch on High with Execute
blerx	ra	e9ba	Branch on Low or Equal with Execute
blrx	ra	ef9a	Branch on Low with Execute
bmrx	ra	ef9a	Branch on Minus with Execute
bnc0rx	ra	e9ca	Branch on Not Carry 0 with Execute
bnerx	ra	e9aa	Branch on Not Equal with Execute
bnhrx	ra	e9ba	Branch on Not High with Execute
bnlrx	ra	e99a	Branch on Not Low with Execute
bnmrx	ra	e99a	Branch on Not Minus with Execute
bnorx	ra	e9ea	Branch on Not Overflow with Execute
bnprx	ra	e9ba	Branch on Not Plus with Execute
bntbrx	ra	e9fa	Branch on Not Test Bit with Execute
bnzrx	ra	e9aa	Branch on Not Zero with Execute
borx	ra	efea	Branch on Overflow with Execute
bprx	ra	efba	Branch on Plus with Execute
brx	ra	e98a	Branch with Execute
btbrx	ra	effa	Branch on Test Bit with Execute
bzrx	ra	efaa	Branch on Zero with Execute
noprx	ra	ef8a	No Operation but with Execute

8.4. Extended Mnemonics: Jump

The operand field consists of a label defined in the same text or data segment as the jump instruction, and located within -256 to +254 bytes.

Source Syntax		Assembled Format	Operation
j	lbl	00dd	Jump
jc0	lbl	0cdd	Jump on Carry 0
je	lbl	0add	Jump on Equal
jeq	lbl	0add	Jump on Equal
jh	lbl	0bdd	Jump on High
jhe	lbl	01dd	Jump on High or Equal
jl	lbl	09dd	Jump on Low
jle	lbl	03dd	Jump on Low or Equal
jm	lbl	09dd	Jump on Minus
jnc0	lbl	04dd	Jump on Not Carry 0
jne	lbl	02dd	Jump on Not Equal
jnh	lbl	03dd	Jump on Not High
jnl	lbl	01dd	Jump on Not Low
jnm	lbl	01dd	Jump on Not Minus
jno	lbl	06dd	Jump on Not Overflow
jnop	lbl	08dd	No Operation
jnp	lbl	03dd	Jump on Not Positive
jntb	lbl	07dd	Jump on Not Test Bit
jnz	lbl	02dd	Jump on Not Zero
jo	lbl	0edd	Jump on Overflow
jp	lbl	0bdd	Jump on Positive
jtb	lbl	0fdd	Jump on Test Bit
jz	lbl	0add	Jump on Zero

8.5. Macro Instructions

The macro instructions generate different instruction sequences depending upon the value of an operand:

sil ra,rb,i

generates an 'ail' with the value of i negated; i must be between −32767 and 32768.

mr ra,rb

generates a 'cas' with r0 as the third operand.

ai ra, [rb,] i
si ra, [rb,] i
ci ra, i
cli ra, i

generates a long or short format instruction depending upon the value of i, and substitutes ra for an omitted rb.

ni ra,rb,i

gives the effect of an and with a 32-bit i by generating a sequence of one or two 'niuz', 'niuo', 'nilz', and 'nilo' instructions.

xi ra,rb,i

gives the effect of an exclusive or with a 32-bit i by generating 'xiu', 'xil', 'xiu' and 'xil', or 'cal' and 'x'.

oi ra,rb,i

gives the effect of an inclusive or with a 32-bit i by generating 'oiu', 'oil', 'oiu' and 'oil', or 'cal' and 'o'.

shl ra,i
shla ra,i

generates a 'sli' or 'sli16', depending on i. i must be in 0-31.

shr ra,i

generates a 'sri' or 'sri16', depending on the value of i. i must be in 0-31.

shra ra,i

generates a 'sari' or 'sari16', depending on the value of i. i must be in 0-31.

get	**ra, reloc**
getha	**ra, reloc**
geth	**ra, reloc**
getc	**ra, reloc**
put	**ra, reloc**
puth	**ra, reloc**
putc	**ra, reloc**

generates a storage reference instruction in long or short form depending on the value of the displacement.

The following macros facilitate generating address constants, and loading and storing in arbitrary memory locations, by exploiting split address relocation. (See *a.out*(5).)

get	**ra, $expr[(rb)]**
getha	**ra, $expr**
geth	**ra, $expr**
getc	**ra, $expr**

If the optional index **(rb)** is present, *as* generates a 'cau' and 'cal'. Otherwise, for an absolute $expr, *as* generates a 'lis', 'cal', 'cal16', or 'cal16' and 'oiu', depending upon the value of expr. For a relocatable or external $expr, *as* generates a 'cal16' and 'oiu'.

load	**ra, expr[(rb)]**
load	**ra, expr[(rb)]**
loadh	**ra, expr[(rb)]**
loadha	**ra, expr[(rb)]**
loadc	**ra, expr[(rb)]**

As generates a 'cau ra' followed by 'l,' 'lh,' 'lha,' or 'lc'. expr may be absolute, relocatable, or external. *ra* may not be r0.

store	**ra, expr[(rb)],rc**
storeh	**ra, expr[(rb)],rc**
storeha	**ra, expr[(rb)],rc**
storec	**ra, expr[(rb)],rc**

As generates a 'cau rc' followed by 'st', 'sth', or 'stc'. expr may be absolute, relocatable, or external. *rc* is a temporary register and may not be r0. storeha is equivalent to storeh.

9. DIAGNOSTICS

Diagnostics are written to standard output. They are intended to be self-explanatory and report errors and warnings. Error diagnostics complain about lexical, syntactic and some semantic errors, and abort the assembly.

The assembler may abandon a statement in error and continue processing sometimes on the same line, sometimes on the next. The result is that one error may lead to spurious diagnostic messages and sometimes "phase errors" where a label has a changed value in the second pass.

10. LIMITS

limit	what
arbitrary[6]	Files to assemble
BUFSIZ	Significant characters per name
arbitrary	Characters per input line
arbitrary	Characters per string
arbitrary	Symbols
4	Text segments
4	Data segments

The number of tokens in a literal definition is limited by the size of the tokenized literal (i.e. by the size of the literal after it has been scanned by the assembler to form a string of tokens). The effective limit is approximately twenty terms in one literal expression.

[6]Although the number of characters available to the *argv* line is restricted by UNIX operating systems to 10240.

Floating Point Arithmetic

Floating point arithmetic on the IBM RT PC conforms to IEEE Standard 754 for binary floating point arithmetic. Single- and double-precision representations are supported; extended is not.

IEEE arithmetic produces results that in general are at least as accurate as those from IBM System/370 arithmetic. Single precision is very similar to VAX F-format in range and precision. Double precision is comparable to VAX D-format; see (1) below.

The salient differences from the F- and D-format arithmetic used in C and 4.2BSD on the VAX are as follows:

(1) Type double has a mantissa of 53 bits rather than 56; the exponent range is approximately 3e-308 to 1e308, rather than 3e-39 to 1e38. Magnitudes as small as 3e-324 are represented with reduced precision.

(2) IEEE arithmetic includes representations for plus and minus infinity and a collection of "Not-a-Number" (NaN) values. *Printf* (3S) represents these on output as INF and NAN(). Signed zero values are also supported; $+0 = -0$, but $1/-0 = -$INF.

(3) Rounding modes and exception handling are supported; user code can change the settings via the swapround, swapfpflag and swapfptrap functions. See *ieee*(3). IEEE default settings are in force initially: the rounding mode is round to nearest; on an exception, proceed without trap (i.e. return a reasonable result).

(4) With the default exception handling, several arithmetic operations that signal SIGFPE on the VAX do not on the IBM RT PC. Exponent overflow receives IEEE default handling, which is to return infinity. Other values larger than 1e38 are represented correctly rather than overflowing. 0/0, INF/INF and certain other operations produce NaNs, which will propagate through subsequent arithmetic operations. Library functions that signaled SIGFPE, however, continue to do so.

(5) VAX F and D formats differ only in mantissa width: the first word in D-format has the same interpretation as an F-format number. Consequently, on a VAX, type mismatches can produce plausible incorrect results, differing from the correct results by one part in a million. IEEE single and double formats differ in exponent width as well as mantissa width, so type mismatches (from nonportable unioning, function calls, or using "%e" for "%le" in *scanf* (3S), for instance) generally produce answers that are dramatically, rather than subtly, wrong.

(6) The IEEE recommended functions are supported; see *ieee*(3) for details.

Also, two new functions are provided to perform the IEEE required operations of round floating-point number to integral value (according to the current rounding mode) and floating-point remainder. These are *rint* (see *floor*(3M)) and *drem* (see *ieee*(3)), respectively.

Floating point operations are performed either by the hardware Floating Point Accelerator (FPA) or by a software FPA emulator. The compiled or ".o" form of a program is suitable for either environment. When .o's are linked into an object program, *ld*(1) generates either a "compatible" or "direct" form, depending on the **-lfpa** flag. By default, *ld* generates an object program that performs floating point arithmetic through the compatibility interface, using the FPA if it is present or the emulator if it is not. With the **-lfpa** option, *ld* generates an object program that presumes the FPA's presence to reduce execution-time overhead. Such a program runs without an FPA present; however, each floating point operation causes a kernel interrupt, making execution very slow.

Compilers also accept the **-lfpa** flag, and pass it on to *ld*. Only the **-lfpa** flag can specify a "direct" object program; the presence of an FPA on the machine performing compilation and linking has no bearing. See *fpa*(3X) and "4.2/RT Linkage Convention" in Volume II, Supplementary Documents.

The C Subroutine Interface for the IBM Academic Information Systems Experimental Display

This paper describes a subroutine interface for the IBM Academic Information Systems experimental display transported for use under the C programming language and 4.2/RT. It contains the following chapters and appendices:

1. INTRODUCTION

The experimental display is a black-and-white, all-points-addressable, bit-mapped display that attaches to the IBM RT PC. The experimental display features 819,200 points on the screen, each one individually selectable. The experimental display adapter contains a very fast on-board processor that allows text and graphics to be drawn at a rate much faster than the host alone would allow. The experimental display processor is programmed to accept high-level orders from the host, and to present the results on the screen.

The characteristics of communicating with the experimental display are determined by the microprogram running in the experimental display adapter processor. This program is stored in writable control store and is loadable from the host.

The interface described in this paper is a set of functions designed to support a window manager, and is composed primarily of subroutines, as distinguished from functions. A typical subroutine uses parameters to receive input as well as to return output. C passes parameters by value; to call a subroutine which returns information, you must supply an address for the returning value as the parameter.

Calls that supply an *address* for return in this package should usually supply the address of a *short* (16-bit) integer. Calls that pass integer *values* can usually get by with either *short* or *int*. See the individual routines.

Many of the subroutines do return a value as a function would. Generally, values are used for error return codes and special case handling. It is strongly recommended that applications monitor return codes in order to prevent bizarre events and possibly more severe errors.

When linking, you must specify *-laed* to pick up the experimental display library.

2. CONTROLLING THE INTERFACE

This chapter describes the subroutines that control the interface.

2.1. VI_Init: Initialize the Subroutine Interface

VI_Init initializes the experimental display and returns the dimensions of the screen. Current display models are 1024 bits wide by 800 bits high. The top left point is (0,0) and the bottom right point is (1023,799). A 16-bit word used as an image on the experimental display will have its least significant bits to the right. */usr/lib/aed/whim.aed* must be accessible at run time.

Because *VI_Init* initializes the experimental display, it should be called before the other routines of the package.

If another user has opened the */dev/aed* device, that user has graphics control of the experimental display and *VI_Init* will fail. For more information, see *ibmaed*(4).

VI_Init has the following format:

```
VI_Init(wd,ht)
  short *wd,*ht;          /* screen dimensions */
```

2.2. VI_Force: Force Output of Graphics Orders

Commands built with subroutines described in "Setting Graphics Parameters" and "Issuing Graphics Primitives" later in this paper generally do not send their output to the screen immediately. Instead the output remains in a buffer until the buffer is full, when its output is sent to the screen. Use *VI_Force* to force output in the current buffer to be transmitted before the buffer is full.

VI_Force has the following format:

```
VI_Force()
```

2.3. VI_Login: Begin Logging Subroutine Calls

VI_Login specifies that subsequent subroutine calls are to be echoed into the specified file. If a log file is already open, *VI_Login* closes it before opening the new file; *VI_Login* overwrites an existing file. All orders to the experimental display are logged until a logout call (*Logout*) is issued. The log file may later be executed from within a program using *VI_Run* or on its own using *aedrunner*(1). It may also be examined with *aedjournal*(1). (Appendix C of this paper describes these programs.) *VI_Login* returns a negative value if there is an error, and a nonnegative value if the call is successful.

VI_Login has the following format:

```
int VI_Login(filename)
    char *filename;  /* file to log to */
```

2.4. VI_Logout: Close a Log File

VI_Logout closes the log file and returns one of three values:

Value	Meaning
0	Normal completion
-1	Error in closing file
-2	No file found to close

VI_Logout has the following format:

```
int VI_Logout()
```

2.5. VI_Run: Process a Log File

VI_Run executes the commands logged in the specified file; *filename* is the name of a log file that was created by *VI_Login*. Using *VI_Run* with a log file has the same effect of executing *aedrunner*(1) from within a program, allowing a series of orders which require much calculation to be figured only once, logged, then quickly retrieved when needed. *VI_Run* returns 0 for a normal completion, and -1 for an error condition.

VI_Run has the following format:

```
int VI_Run(filename)
    char *filename;        /* log file name */
```

2.6. VI_Term: Terminate the Subroutine Interface

VI_Term completes processing, closes the log file, and forces transmission of the graphics buffer to the experimental display.

VI_Term has the following format:

```
VI_Term()
```

3. SETTING GRAPHICS PARAMETERS

Graphics parameters modify the way in which the primitives described later in this paper operate. This chapter describes the subroutines that set graphic parameters. The initial values of these parameters are:

Clipping window	The clipping window is set to the whole screen.
Screen color	The screen color is white 1's on black 0's, color 0.
Dash pattern	The line dash pattern is solid 1's.
Font	The font is 0. No font is selected.
Merge mode	The merge mode is 12, for replace mode. Data bits replace screen bits.
Line width	Line width is 1.

3.1. VI_Clip: Set Clipping Window

VI_Clip specifies that subsequent primitives drawn on the screen are to be clipped to the specified area. It is the user's responsibility to ensure the sensibility of the window definition.

VI_Clip has the following format:

```
VI_Clip(lx,ly,hx,hy)
 int lx,ly;   /* top left corner of clipping area */
 int hx,hy;    /* bottom right corner of area */
```

3.2. VI_Color: Change Screen Color

VI_Color sets the color of the screen to the specified value: 0 means that bits having the binary value "0" will be black on the screen; 1 means that bits having the binary value "1" will be black on the screen. If this value is different from the previous value, the screen will be inverted, so as to make the change transparent to the application.

VI_Color has the following format:

```
VI_Color(color)
 int color;      /* new color, true for white */
```

3.3. VI_Dash: Set Line Dash Pattern

If no dash pattern has been set, lines drawn with the *VI_RLine* and *VI_ALine* subroutines described in "Issuing Graphics Primitives" are solid lines of 1's. If a pattern has been set, the bits of the pattern word are used in sequence whenever the vector generator would normally output a 1. Setting a pattern of 0x5555 produces a very acceptable dotted line. Other patterns may be used to vary the size of dashes in the line. The length of the pattern can range from 1 to 16 bits. The pattern bits should be left-justified. Setting the pattern length to 0 specifies a return to solid lines.

VI_Dash has the following format:

```
VI_Dash(dash,dashlen)
 unsigned short dash;        /* dash pattern */
 short dashlen;       /* dash pattern length */
```

3.4. VI_Font: Select Font

The current font affects the results of the *VI_String* primitive described under "Issuing Graphics Primitives." Font IDs range from 0 to 255 and are returned by calls to *VI_GetFont*. See "Defining Fonts" later in this paper for more information.

VI_Font has the following format:

```
VI_Font(fontid)
 int fontid;             /* font ID */
```

3.5. VI_Merge: Set Merge Mode

The merge mode is a number from 0 to 15 that specifies how the bits generated by primitives are to be combined with bits already on the screen. The merge mode is simply an encoding of the logical function used to combine screen bits and data bits. Encoding the desired result of each of the combinations in the table below generates the merge mode that should be used to get that effect. For example, to *or* the data you are adding with the data already on the screen, you would use a merge mode of 14:

Data Bit	1	1	0	0	
Screen Bit	1	0	1	0	
Example: OR mode	1	1	1	0	= 14

VI_Merge has the following format:

```
VI_Merge(merge)
 int merge;              /* merge mode */
```

3.6. VI_Width: Set Line Width

VI_Width specifies a value between 1 and 16 that is to be the line width. Normally, lines are 1 bit thick.

VI_Width has the following format:

```
VI_Width(width)
 int width;              /* line width */
```

4. QUERYING GRAPHICS PARAMETERS

The subroutines in this chapter return the current values of the graphics parameters described above. Each subroutine requires an address in which to store the value to be returned. All of these subroutines force transmission of graphics data in the current buffer.

4.1. VI_QClip: Query Clipping Rectangle

VI_QClip returns the current clipping rectangle.

VI_QClip has the following format:

```
VI_QClip(lx,ly,hx,hy)
 short *lx,*ly;      /*top left corner of clipping area*/
 short *hx,*hy;         /* bottom right corner */
```

4.2. VI_QColor: Query Current Color

VI_QColor returns the current color of the screen: 0 means that bits having the binary value "0" will be black on the screen; 1 means that bits having the binary value "1" will be black on the screen.

VI_QColor has the following format:

```
VI_QColor(color)
 short *color;  /* current color, true for white */
```

4.3. VI_QDash: Query Dash Pattern

VI_QDash returns the current line dash pattern in the format described for *VI_Dash*. If *dashlen* is 0, the lines are solid.

VI_QDash has the following format:

```
VI_QDash(dash,dashlen)
 unsigned short *dash;        /* dash pattern */
 short *dashlen;      /* length of dash pattern */
```

4.4. VI_QFont: Query Font

VI_QFont returns the ID and name of the current font. The font ID is 0 if no font has been set. The pointer *fontname* should point to a block of characters large enough to hold a file name (including an extension) on your operating system, along with a string-termination byte. If you know beforehand the size of your file name, you may allow only as many bytes as required. Be aware of the string-terminator byte; there must be room for it.

VI_QFont has the following format:

```
VI_QFont(fontid,fontname)
 short *fontid;             /* current font ID */
 char *fontname;      /* current font name */
```

4.5. VI_QMerge: Query Merge Mode

VI_QMerge returns the current merge mode in the format described for the *VI_Merge* subroutine described in "Setting Graphics Parameters."

VI_QMerge has the following format:

```
VI_QMerge(merge)
 short *merge;          /* current merge mode */
```

4.6. VI_QPoint: Query Current Point

VI_QPoint returns the location of the current point. This command is especially useful after a *VI_String* primitive has been issued, since character definitions can change the current point in unpredictable ways.

VI_QPoint has the following format:

```
VI_QPoint(x,y)
 short *x,*y;              /* current point */
```

4.7. VI_QWidth: Query Line Width

VI_QWidth returns the current line width as a number between 1 and 16.

VI_QWidth has the following format:

```
VI_QWidth(width)
 short *width;             /* line width */
```

5. ISSUING GRAPHICS PRIMITIVES

This chapter describes the subroutines that build orders that update the screen. Orders are transmitted only when the buffer is full, when specified with *VI_Force*, or when other non-graphics subroutines are called.

The graphics primitives work in screen coordinates: *x* represents the horizontal axis on the screen, and increases to the right; *y* represents the vertical axis and increases to the bottom of the screen. The coordinates (0,0) represent the top-left corner of the screen. Subroutines will accept coordinates that are off the screen; the behavior is as if there were a clipping window the size of the screen in a larger universe.

Several of the primitives depend on the current point. This point is initially set to (0,0) and can be modified by primitives.

5.1. VI_AMove: Move the Current Point to an Absolute Location

VI_AMove moves the current point to the specified coordinates. No change is made to the screen.

VI_AMove has the following format:

```
VI_AMove(x,y)
 int x,y;               /* new point */
```

5.2. VI_RMove: Move the Current Point to a Relative Location

VI_RMove moves the current point by the specified displacement. No change is made to the screen.

VI_RMove has the following format:

```
VI_RMove(dx,dy)
 int dx,dy;    /* displacement from old point */
```

5.3. VI_ALine: Draw a Line with an Absolute Location

VI_ALine draws a line from the current point to the specified point (the line's end point) according to the current values of the width and dash pattern parameters. A line is normally of 1's, and is merged with the window data according to the current merge mode. The specified point becomes the current point.

VI_ALine has the following format:

```
VI_ALine(x,y)
 int x,y;              /* end point of line */
```

5.4. VI_RLine: Draw a Line with a Relative Location

VI_RLine draws a line from the current point to the current point displaced by the specified values, according to the current values of the width and dash pattern parameters. A line is normally of 1's, and is merged with the window data according to the current merge mode. The current point is incremented by the displacement.

VI_RLine has the following format:

```
VI_RLine(dx,dy)
 int dx,dy;       /* displacement to endpoint */
```

5.5. VI_Circle: Draw a Circle

VI_Circle draws a circle with the specified radius and the current point as its center. The current point is unchanged.

VI_Circle has the following format:

```
VI_Circle(radius)
 int radius;              /* circle radius */
```

5.6. VI_MImage: Draw an Image from Memory

VI_MImage draws an image of the specified dimensions whose top left corner is at the current point. The current point is not changed.

Data must be the first byte of an image large enough to fill the rectangle specified by *wd* and *ht*, or an addressing error may result. The image data should be in scanline order, from top to bottom, with each scanline padded to the next 16-bit word. For example, for a width of WD and height of HT, there should be 2*HT*(WD+15)/16 bytes of image data.

VI_MImage has the following format:

```
VI_MImage(wd,ht,data)
 int wd,ht;           /* dimensions of image */
 unsigned short *data; /* first byte of image */
```

5.7. VI_FImage: Draw an Image from a File

VI_FImage draws the image contained in the specified file, placing its top left corner at the current point. The current point is unchanged.

The image file must have the format shown below. The data words should be in the same format as for the *VI_MImage* subroutine.

Offset (bytes)	Description
0	The width of the image
2	The height of the image
4	Image data

VI_FImage has the following format:

```
VI_FImage(filename)
 char *filename;     /* file name of image to draw */
```

5.8. VI_Tile: Tile a Rectangle

VI_Tile fills a rectangle of the specified dimensions with the specified pattern. The rectangle's top left corner will be at the current point. The tile pattern must follow the rules for images (see the *VI_MImage* subroutine above), and can be of any size. The tile pattern is aligned to multiples of *twd* and *tht*, not to the bounds of the tiled rectangle, so that rectangular subareas of larger figures can be tiled without regard to their bounds, and the tile patterns will match. The current point is unchanged.

A full rectangle black or white fill can be most quickly drawn by requesting a one-by-one tile. Clearly, only all ON or all OFF may be drawn with this method, but any merge mode may be used.

VI_Tile has the following format:

```
VI_Tile(wd,ht,twd,tht,tile)
 int wd,ht;         /* dimensions of rectangle */
 int twd,tht;            /* dimensions of tile */
 unsigned short *tile;  /* first byte of pattern */
```

5.9. VI_String: Draw a String

VI_String draws the specified string at the current point. Since a character definition is really a sequence of other graphics commands (usually *VI_MImage* and *VI_RMove*), the way in which characters are positioned, stepped, and drawn depends on the font definition. Character definitions typically modify the current point. See "Defining Fonts" later in this paper for more information.

VI_String has the following format:

```
VI_String(s)
 char *s;              /* string to draw */
```

5.10. VI_Copy: Copy an Area

VI_Copy duplicates the rectangle at *sx,sy* with the dimensions *wd,ht* to the point *tx,ty*. The copied bits are merged with the target area using the specified merge mode, not the merge mode set by *VI_Merge*.

Both the source and destination rectangles must be completely on the screen. The current setting of the clipping window is ignored.

VI_Copy has the following format:

```
VI_Copy(sx,sy,tx,ty,wd,ht,merge)
 int sx,sy;            /* source top-left */
 int tx,ty;            /* target top-left */
 int wd,ht;            /* rectangle dimensions */
 int merge;            /* merge mode */
```

5.11. VI_MRead: Read Display Data into Memory

VI_MRead reads the specified area of the screen into the array passed as *data*. Image bytes are in the same format as expected by *VI_MImage*. If the screen color is white, the bits are inverted on readback to make the data read back independent of screen color. The area to be read must be completely on the screen. The current setting of the clipping window is ignored.

VI_MRead has the following format:

```
VI_MRead(x,y,wd,ht,data)
 int x,y;              /* top-left corner of area */
 int wd,ht;            /* dimensions of area */
 unsigned short *data; /* first byte of data */
```

5.12. VI_FRead: Read Display Data into a File

VI_FRead reads the specified area of the screen and places it in the specified file. The file has the same format as expected by *VI_FImage*. If the window color is white, data bits are inverted to make the data independent of the screen color. The area to be read must be completely on the screen. The current setting of the clipping window is ignored.

VI_FRead has the following format:

```
VI_FRead(x,y,wd,ht,filename)
 int x,y;              /* top-left corner of area */
 int wd,ht;            /* dimensions of area */
 char *filename;       /* name of file to place image in */
```

6. CONTROLLING THE CURSOR

The following routines allow programs to control the experimental display cursor by defining it, enabling and disabling it, and changing its position. Note that because the experimental display maintains the cursor separately from the display buffer, the cursor does not have to be removed when a graphics primitive intersects its position.

Initially the cursor is transparent and disabled, and is positioned at the center of the screen.

6.1. VI_MDefnCur: Set Cursor Pattern from Memory

VI_MDefnCur sets the cursor as specified. *xoff,yoff* is the displacement of the cursor pattern from the current position of the cursor. For example, a value of (32,32) would center the cursor pattern around the current point.

The cursor pattern itself is a 64-by-64 bit image, with two planes. A 1 in the black plane indicates that that bit of the cursor should be black. A 1 in the white plane indicates that the cursor should be white in that position. If a bit has a 0 in both planes, the cursor is transparent in that position. If a bit is 1 in both planes, the cursor is white.

The two planes are images in the same format as accepted by *VI_MImage*, and must be 64-by-64, or 512 bytes each.

VI_MDefnCur has the following format:

```
VI_MDefnCur(xoff,yoff,black,white)
 int xoff;        /* x offset of cursor center */
 int yoff;        /* y offset of cursor center */
 unsigned short *black;    /*first byte black mask */
 unsigned short *white;    /*first byte white mask */
```

6.2. VI_FDefnCur: Set Cursor Pattern from File

VI_FDefnCur sets the cursor to the definition in the specified file. The file has the following format:

Offset (bytes)	Description
0	XOFF
2	YOFF
4	BLACK bit pattern
516	WHITE bit pattern

See the description of *VI_MDefnCur* for a description of the fields.

VI_FDefnCur has the following format:

```
VI_FDefnCur(filename)
 char *filename;     /* name of cursor definition file */
```

6.3. VI_EnCur: Enable Cursor

VI_EnCur enables the cursor and displays it if it is not already present. Disabling and reenabling the cursor do not affect its position.

VI_EnCur has the following format:

```
VI_EnCur()
```

6.4. VI_DisCur: Disable Cursor

VI_DisCur disables the cursor and removes it from the screen if it is present. Disabling and reenabling the cursor do not affect its pattern or position.

VI_DisCur has the following format:

```
VI_DisCur()
```

6.5. VI_PosnCur: Set Cursor Position

VI_PosnCur moves the cursor to the specified position. The cursor cannot be moved off the screen.

VI_PosnCur has the following format:

```
VI_PosnCur(x,y)
 int x,y;            /* new cursor position */
```

7. DEFINING FONTS

The font mechanism supported by the experimental display is very general. Characters are not simply raster patterns; instead, each character definition is a simple graphics subroutine, able to move the current point, draw images, change the merge mode, etc. The orders that can occur in a character definition are a subset of the orders built by the graphics primitives subroutines. In addition, two orders, *push* and *pop,* control parameters within a character definition.

7.1. Standard Raster Characters

The most typical use of the font mechanism is for standard raster characters. The sequence of orders is similar to the following:

(1) *VI_Image* at the current point.

(2) *VI_RMove* right by the width of the characters.

This example draws all characters down from the current *y* value.

7.2. Raster Character with Baseline Defined for the Font

The next most common use is a raster character with a baseline defined for the font. The sequence of orders would be similar to the following:

(1) *VI_RMove* up by the ascender height (height above baseline).

(2) *VI_Image* at the current point.

(3) *VI_RMove* down and right by the ascender height and character width.

7.3. Stroked Fonts

Stroked fonts can be defined using *VI_RMove* and *VI_RLine* commands. Stroked characters can be mixed freely with raster characters.

7.4. Three-Color Characters

Three-color characters can be defined with a sequence such as the following:

(1) *VI_RMove* to top of character image.

(2) *VI_Merge 2*, which turns off the screen data having the binary value "1", and leaves it unchanged for screen data having the binary value "0".

(3) *VI_Image*, with a pattern that turns off the black bits of the character.

(4) *VI_Merge 14*, OR mode.

(5) *VI_Image*, with a pattern that turns on the white bits.

(6) *VI_RMove* to start of next character.

With this font selected, characters drawn by the *VI_String* command would draw black, white and transparent patterns, suitable for text drawn over a complex graphics image.

8. MANIPULATING FONTS

Fonts are stored in files, which are loaded into the IBM RT PC memory when requested by applications using the *VI_GetFont* subroutine. Once a font is loaded, it is kept in memory until the program ends, unless explicitly dropped with the *VI_DropFont* subroutine.

8.1. VI_GetFont: Load a Font into Memory

VI_GetFont loads the specified font into memory, if it is not already present. If the font is successfully loaded, the font ID is returned. Setting the current font to this ID with the *VI_Font* routine causes subsequent strings to be displayed in the font. If a font ID of 0 is returned, either the font could not be found, or it did not fit in memory. If the font did not fit in memory, a message will be sent to *stderr*.

VI_GetFont has the following format:

```
VI_GetFont(name,fontid)
 char *name;            /* font name */
 short *fontid;         /* font ID */
```

8.2. VI_DropFont: Release Font

VI_DropFont drops the specified font from memory. The application should not attempt to use the font ID again. If the font is required, a new font ID should be generated by a request to *VI_GetFont*.

VI_DropFont has the following format:

```
VI_DropFont(fontid)
 int fontid;        /* ID of font to release */
```

APPENDIX A. FORMAT OF A FONT FILE

A font definition file begins with an index by character codepoint. The first entry is for codepoint 0x00, the second for 0x01, and so on, up to 0xFF. An index entry has the following format:

Offset	Length in bytes	Description
0	4	Offset of the character definition in the file; an undefined character has an offset of zero.
4	2	Width of inner box of the character.
6	2	Height of inner box of the character.
8	2	Total x displacement caused by character.
10	2	Total y displacement caused by character.
12	2	Distance from the initial x position to the left edge of the inner box.
14	2	Distance from the initial y position to the top edge of the inner box.

A font file consists largely of character definitions, which follow the index. Character definitions do not necessarily appear in order. Undefined characters are not included. Each character definition has the following format:

Offset	Length in bytes	Description
0	2	Character codepoint, in the low byte of the word.
2	2	Length of character definition, in 16-bit words, not including the count. The length of a character definition must be less than 2000 words.
4	count*2	Character definition. A definition consists of a series of orders, as described in Appendix B of this article.

APPENDIX B. CHARACTER DEFINITIONS

Before reading this, you should understand the format of the font file, which contains character definitions, described in Appendix A of this article.

Character definitions consist of a string of orders from the following list. Note that parameter changes made by character definitions do not persist after the character has been completed.

Set Merge Mode

Offset in 16-bit words	Value
0	Merge Command (= 1)
1	Merge Mode

The merge mode is changed to the specified value. The format is the same as described for the *VI_Merge* subroutine.

Set Line Dash Pattern

Offset in 16-bit words	Value
0	Set Dash Command (= 3)
1	Dash Pattern
2	Pattern length

Lines drawn after this command use the specified pattern. A pattern length of zero specifies a return to normal solid lines. The pattern is from 1 to 16 bits, left-justified in the pattern word.

Set Line Width

Offset in 16-bit words	Value
0	Set Width Command (= 4)
1	Line Width

Subsequent lines are drawn with the specified width.

Push Modes

Offset in 16-bit words	Value
0	Push Command (= 12)

The modifiable parameters (merge mode, dash pattern, line width) are pushed onto an internal stack. They may be changed and then later restored with the *pop* order. When a character definition ends, the original modes are restored, regardless of *push* or *pop* orders within a definition.

Pop Modes

Offset in 16-bit words	Value
0	Pop Command (= 13)

The modifiable parameters (merge mode, dash pattern, line width) are restored from the internal stack. When a character definition ends, the original modes are restored, regardless of *push* or *pop* orders within a definition.

Move Relative

Offset in 16-bit words	Value
0	Move Relative Command (= 6)
1	X displacement
2	Y displacement

The indicated displacement is added to the current point. If either coordinate of the current point goes outside the range -32768 to 32767, the value wraps (overflows or underflows).

Draw Line Relative

Offset in 16-bit words	Value
0	Draw Line Relative Command (= 8)
1	X displacement
2	Y displacement

A line is drawn from the current point to the current point plus the displacement. The ending point becomes the new current point. If either coordinate of the current point goes outside the range -32768 to 32767, the value wraps (overflows or underflows).

Draw Circle

Offset in 16-bit words	Value
0	Draw Circle Command (= 14)
1	Circle Radius

A circle with the specified radius is drawn around the current point. The current point is unchanged.

Draw Image

Offset in 16-bit words	Value
0	Draw Image Command (= 9)
1	Image width
2	Image height
3	Image data

The image given is drawn with its top left corner at the current point. The current point is unchanged.

The scanlines of the image must be padded to the next 16-bit word. Thus, the number of words in the image is height*(width + 15) / 16.

Tile Rectangle

Offset in 16-bit words	Value
0	Tile Command (= 10)
1	Rectangle width

2	Rectangle height
3	Tile width
4	Tile height
5	Tile data

The tile image is repeated over the whole area of the indicated rectangle. The tile image data has the same format as data in the *VI_Image* order described above.

APPENDIX C. AEDJOURNAL AND AEDRUNNER

Aedjournal(1) and *aedrunner*(1) are supplied programs which use the interface. Both operate on a log file created with *VI_Login* and *VI_Logout*. *Aedjournal* displays the commands built into the file; *aedrunner* executes those commands.

Debugging with aedjournal

```
aedjournal file
```

Although there is no debugging facility as such supplied with this package, you can use *VI_Login* and *VI_Logout* with *aedjournal* to help follow your application program's actions. *Aedjournal* deciphers a file produced by *VI_Login* and reports to standard output all orders passed to the experimental display. Standard output may be redirected as usual. You may inspect this output to discover unintended results.

Beware of the length of logged files. It is very easy to generate thousands of display orders for a seemingly simple picture; thus, try to log the smallest group of orders you believe contains the error. The log routines may be called several times in one application to produce several files of orders, requiring only that each call to *VI_Login* provide a distinct file name.

Executing a log file with aedrunner

```
aedrunner file ...
```

Aedrunner executes the orders logged into the specified file, which must have been created with *VI_Login* and *VI_Logout*. *Aedrunner* terminates upon discovery of any error or inconsistency in the file. All additional files which were needed when the log file was constructed must be available in the current directory. Such files are any font, image, or cursor definition files you may have used, and */usr/lib/aed/whim.aed* must exist. Images, cursors, or tiles defined from memory are handled by the log routines and do not require regeneration.

APPENDIX D. SUPPLIED EXAMPLES

All files associated with this package reside in the directory */usr/src/usr.lib/libaed/examples.*

Among the files supplied with the microcode and subroutine library are some source and executable files for you to investigate. The following list includes some of those files, and brief descriptions. It should be easy to figure out the nature of any other files from their names, behavior, or above documentation.

The following programs are copyrighted property of International Business Machines Corporation.

*.fnt	Files with the extension *.fnt* are font files.
showfont	A program that shows a font on the experimental display. The syntax is *showfont filename.*
showfont.c	Source for *showfont.*
zip	A demo that takes up to three parameters. Parameter 1 is number of vectors to remain on the screen. Parameter 2 is minimum delta for each new vector endpoint. It is roughly equivalent to the speed of the zipper. Parameter 3 is maximum delta. The default is *zip 30 2 14.*
zip2	Like *zip* but with two zippers. It takes up to 6 parameters. The default is *zip2 30 2 14 90 1 4.*
zipn	Like *zip* but with 1 to 16 zippers. Parameter 1 is number of zippers. Parameters 2, 3, and 4 are number vectors for zipper 1, minimum delta, and maximum delta. Parameters 5, 6, and 7 are for zipper 2, etc. The default for unspecified zippers is *30, 2,14.* The default is *zipn 1.*
zip.c	Zip source code.
aedrunner.c	*Aedrunner* source code.

Programmer's Notes

1. INTRODUCTION

This article is a compendium of insights, suggestions, and notes gathered from the programmers who ported applications to 4.2/RT. The information could save you time and frustration as you port programs to operate under 4.2/RT.

2. SAMPLE FILES PROVIDED

Four sample files (.login, .cshrc, .logout, and .profile) are provided in /usr/skel. Using these files will simplify initial installation and operation of 4.2 on the IBM RT PC.

3. CHARACTER TYPE IS UNSIGNED

Variables of type **char** are unsigned (range 0..255) by default on the RT PC, in contrast to the VAX, where they are signed (range -128..127) by default. With the High C compiler (*hc*(1)), the type **signed char** is available, as well as a command-line option *-Hoff=char_default_unsigned* to make characters signed by default. This option generally produces less efficient code, but can be of value in determining whether signedness is the cause of a bug.

The **unsigned** default uncovers a machine dependency in a common technique for end-of-file testing. In the following program fragment

```
char c;
...
if ((c = getchar()) == EOF) ...
```

the test always fails, since EOF is -1 and c is in 0..255. Declaring c as an **int** is a good machine-independent solution.

With *pcc*(1), there is no type **signed char**, but the following macro might be useful if you need to use an unsigned character as though it were signed:

```
#if '\377' < 0
#define Signed(x) (x)
#else
#define Signed(x) (((x)^128)-128)
#endif
```

4. BYTE ORDERING IS DIFFERENT

The IBM RT PC has sixteen 32-bit general registers. Memory on the IBM RT PC is byte-addressed, but differently than on the VAX.

On the VAX, high order bits are at higher addresses, thus:

```
|---word2---|---word1---|---word0---|
|C3,C2,C1,C0|C3,C2,C1,C0|C3,C2,C1,C0|
|B31......B0|B31......B0|B31......B0|
```

On the IBM RT PC, high order bits are at lower addresses, thus:

```
|---word0---|---word1---|---word2---|
|C0,C1,C2,C3|C0,C1,C2,C3|C0,C1,C2,C3|
|B0......B31|B0......B31|B0......B31|
```

Non-portable code which depends upon byte ordering for retrieving data must be rewritten.

5. ALL MEMORY REFERENCES ARE ALIGNED

Word and half-word data are stored most significant byte first and aligned on natural boundaries. Off-boundary storage references are not supported. The low two or one address bits are silently ignored, creating unexpected results.

If *lint*(1) is run against such programs, it complains about a "possible alignment problem."

6. FLOATING POINT IS IEEE STANDARD

4.2/RT conforms to IEEE Standard 754 for binary floating point arithmetic. The article "Floating Point Arithmetic" in Volume II notes the differences from VAX floating point.

A class of programming errors easily overlooked on the VAX -- treating the first half of a **double** quantity as a **float** quantity, or vice versa -- is highly visible on the RT PC. If numeric results are incorrect, look first for unions, casts, or function arguments that mismatch **double** and **float**. The *scanf* format "%f" instead of "%lf" is particularly subtle.

7. OLD CALLING SEQUENCE IS NO LONGER SUPPORTED

The subroutine calling sequence currently used in 4.2/RT first appeared in the March 1986 release. As a transition aid, that release also supported the old calling sequence.

The December 1986 release (PRPQ #5799-CGZ, Release 2) supports only the current calling sequence. In the unlikely event that your installation still has programs not recompiled since you installed the March release, you must recompile and relink them. In the current release, running an old a.out will produce the message: **old calling sequence**, then terminate.

In the even more unlikely event that the source for the old program is no longer available, you can reinstate support for the old calling sequence in the current release (with a performance penalty) by specifying "option DUALCALL" in the kernel config file and rebuilding the kernel. See the article "Building 4.2/RT Systems with Config" in Volume II.

Some of the IBM Support tools provided in the March release used the old calling sequence. Be sure to replace these by the versions provided in the December release.

8. CAUTION WHEN USING THE 4.3 AT COMMAND

The 4.3 *at*(1) command does not pass the environmental variable **TERM** into a user's *at* spool file. Spool-file processing may break if the user's *.cshrc* file includes a reference of the form "$TERM" and the user's environmental shell is csh. To be defensive, csh users should code their *.cshrc* files in such a way as to test whether a variable is set before being referenced. For example:

```
if ($?TERM) then                    # is TERM defined?
        if ($TERM = = h19) then
                setenv MORE -c
        endif
endif
```

(This is good programming practice for *.login* files as well).

9. CAUTION WHEN USING THE 4.3 CSH ON SETUID SCRIPTS

The 4.3 *csh*(1) command requires that a **-b** flag be used on the interpreter line of setuid *csh* scripts. *Csh* exits with a "Permission denied" error message if the **-b** flag is not specified.

The IBM 3812 Pageprinter

ABSTRACT

This article provides information for using the IBM 3812 Pageprinter[7] attached to an IBM RT PC, and installing and converting fonts.

The article contains the following chapters and appendices:

1. **Introduction** provides general information about the 3812.

2. **Printer Installation** explains how to install the 3812 on an IBM RT PC running 4.2/RT.

3. **Printing Text Files** describes *pprint*(1), a command used to print text files on the 3812.

4. **Printing Ditroff Files** describes how to print ditroff files on the 3812. Ditroff (device-independent troff) support for the IBM 3812 Pageprinter[8] is a separately-licensed feature of 4.2/RT. It is a modified version of the Documenter's Workbench[9] ditroff.

5. **Fonts** describes how to install both the uniformly-spaced fonts provided with 4.2/RT and the separately-orderable fonts available from IBM.

6. **Using Code Page Tables** gives information about generating fonts for use with the 3812.

Appendix A. Fonts Available on the 3812 lists the IBM fonts that are available for printing with the 3812.

Appendix B. Code Page Tables lists the characters in each code page.

[7] Hereinafter referred to as "the 3812".

[8] Hereinafter referred to as "ditroff."

[9] Documenter's Workbench is a registered trademark of AT&T Technologies.

1. INTRODUCTION

This article provides information helpful in installing and using the 3812 on the IBM RT PC. The 3812 is a multifunction, nonimpact, tabletop page printer. It provides cut-sheet, letter-quality text and all-points-addressable graphics at a maximum rate of 12 pages per minute and a resolution of 240 points per inch. The 3812 attaches to an asynchronous port on an IBM RT PC. The printer can serve users on that machine or other machines on the network. The standard spooling system commands are used to send files to the printer, query their status, and cancel them (see *lpr*(1), *lprm*(1), *lpq*(1), and *lpc*(8)).

Several manual pages describe support for the 3812:

ap(4)	Defines the asynchronous line protocol that supports the 3812
cvt3812(8)	Converts IBM 3820 and 3800 fonts to 3812 format
font3812(5)	Defines the 3812 font structures
ibm3812pp(8)	Describes the 3812 print server daemon
pic(1)	Draws simple pictures on the 3812
pprint(1)	Prints text files on the 3812
ppt(8)	Describes the spooling system filter
printer3812(5)	Defines printer status information
ptroff(1)	Prints ditroff files on the 3812
width3812(8)	Builds table widths for 3812 fonts

2. PRINTER INSTALLATION

2.1. Hardware Requirements

To connect the 3812 to an IBM RT PC, you must have the following:

- A 3812 with an IBM 3812 PC diskette
- An IBM RT PC running 4.2/RT
- An RS232 4-port asynchronous adapter (in the IBM 6151 only)
- An IBM 3812 Printer Cable, part number 1348421 and an IBM RT PC Modem Cable, part number 6298240

  OR
- An IBM IBM RT PC Printer cable, part number 6298525, option number 6294803

  (Note that the single IBM IBM RT PC Printer Cable replaces both the IBM 3812 Printer Cable and the IBM RT PC Modem Cable.)

2.1.1. 3812 Setup

This section describes 3812 setup.

2.1.1.1. DIP Switches

On the inside of the back panel of the 3812 is the set of printer DIP switches. To run at 19200 baud, switches three (3) and seven (7) must be on; the rest must be off.

3 = asynchronous data communications mode

7 = 19200 baud

To run at other than 19200 baud, consult the *IBM 3812 Pageprinter Programming Reference* (S544-3268) for proper DIP switch settings.

2.1.1.2. 3812 Diskette

Inside the front cover of the IBM 3812 Pageprinter is a diskette drive. Load the diskette labeled "IBM 3812 PC"or "IBM 3812 RT" into the drive.

2.1.2. 3812 to IBM RT PC Connection

If you are using two separate cables, follow these steps:

(1) Connect the 3812 Printer Cable to the plug labeled "RS232C" in the back of the 3812.

(2) Connect the other end of the 3812 Printer Cable to the Modem Cable.

(3) Connect the Modem Cable to one of the serial ports on the back of the IBM RT PC.

If you are using the single IBM RT PC printer cable, follow these steps:

(1) Connect the IBM RT PC printer cable to the plug labeled "RS232C" in the back of the 3812.

(2) Connect the other end of the cable to one of the serial ports on the back of the IBM RT PC.

Throughout this article, we will assume the cable is connected to the port corresponding to */dev/tty00.*

2.2. Software Requirements

This section describes the steps required to identify the 3812 to the system.

2.2.1. Verifying Device Files and Entries

Verify that the special file for the printer (*/dev/tty00*) is read/write and owned by daemon. To do so, use the following command:

```
chown daemon /dev/tty00
ls -l /dev/tty00
```

The result should look like this:

```
crw------- 1 daemon    1,   0 Feb  3 15:05 /dev/tty00
```

Verify that the entry for tty00 in the file */etc/ttys* (*ttys*(5)) has as the first character an ASCII zero (0). To do so, use the following command:

```
grep tty00 /etc/ttys
```

The result should be of the form:

```
02tty00
```

If the result you get is, for example, *12tty00*, you must edit the file to change that line to *02tty00.*

2.2.2. Establishing a Symbolic Link

Establish a symbolic link from */dev/pp* to */dev/tty00*. To do so, use the following command:

```
ln -s /dev/tty00 /dev/pp
```

2.2.3. Verifying Printcap Entries

To identify the 3812 to the spooling system, the proper entries must be in */etc/printcap*. As distributed, 4.2/RT has the entries described here; you may need to modify them for local use.

Three sample printcap entries are shown below. The first defines a 3812 attachment on a remote IBM RT PC. The name of the remote host must be substituted for the words *remote 3812 print server name*. The second entry defines a 3812 attachment on the local IBM RT PC. The third entry selects one of the two previous entries. This must be changed to refer to the local entry when installing the 3812.

```
#
# Sample Printcap Entries for 3812
#
# Sample for Remote 3812 Pageprinter
#
rpp|remote 3812 Pageprinter:\
        :lp=:rm=remote 3812 print server name:rp=pp:sd=/usr/spool/rppd
#
# Sample for Locally Attached 3812 Pageprinter, connected at /dev/pp
#
lpp|local 3812 Pageprinter:\
        :lp=/dev/null:sd=/usr/spool/ppd:\
        :lf=/usr/adm/ppd-errs:\
        :af=/usr/adm/acct-pp:\
        :PP=/dev/pp:\
        :SS=/usr/spool/ppd/status3812:\
        :br#19200:\
        :sh:\
        :if=/usr/lib/p3812/txt3812:\
        :vf=/usr/lib/p3812/pmp3812:\
        :pl#66:pw#80:px#2040:py#2460
#
# Change the rpp to lpp in the next line if the 3812 is attached locally
#
pp|3812 Pageprinter:\
        :tc=rpp:
```

Refer to *printcap*(5) for a complete description of the printcap file.

The spool directory (*sd*), accounting file (*af*), and log file (*lf*) must be created read/write by daemon. To do so, issue the following commands:

cd /etc
/usr/src/etc/printcap.install

Note that the filename at the end of the path in the accounting file entry (*af*) must end with a "-" followed by the printer name. In this example, the name is "pp".

Two new printcap capabilities, SS and PP, are defined for the 3812. The SS entry identifies the printer log file for the 3812. The current status message from the printer will be written to this log. It will contain the intervention-required messages such as "PAPER JAM," "OUT OF PAPER,"or "TONER LOW." A list of printer messages can be found in *printer3812*(5).

The PP entry identifies the special file where the 3812 is attached. It is usually a symbolic link to a specific tty port on the multiport adapter card. This information is not provided by the line printer (*lp*) entry. The spooling system does not open or close the device where the printer is attached. Instead, the device is managed by the 3812 print server (*ibm3812pp*(8)), using a special asynchronous line protocol (*ap*(4)) that supports the 3812 printer. The *ap* line discipline is conditionally compiled and must be specified in the kernel configuration file if the 3812 is to be supported. The 3812 print server runs as a daemon and is started by *ppt*(8) when the first file is sent to the printer.

The *sh* entry indicates that spooling system header pages are suppressed. The individual filters (*if*, *vf*) will print header pages if requested.

The following filters are specified in this printcap entry: the text filter (*if*) converts ASCII data to printer commands; the PMP filter (*vf*) sends Page Map Primitive (PMP) commands to the printer. PMP commands provide for vector graphics, font downloading, and all-points-addressable printing on the 3812. PMP is described in the manual entitled *IBM 3812 Pageprinter Programming Reference*, S544-3268.

2.3. Starting the 3812

Once you have completed the steps described thus far, you are ready to print your first document. Turn the printer on and check that paper is loaded in both paper cassettes. The 3812 uses the alternate paper cassette to print document separator pages; using colored paper here will make output separation easier. After a few minutes, a '701' will appear on the printer display, indicating that the printer is ready.

The print server (*ibm3812pp*(8)) starts when the first job is sent to be printed. To start the print server, for example, you can print a copy of */etc/printcap*:

```
pprint /etc/printcap
```

The '701' disappears from the printer display, the online light appears, and the document is printed.

2.4. Monitoring the 3812

The current status of the printer is recorded in the spool directory in the *status3812* file (SS in */etc/printcap*). The following command displays the current status:

```
cat /usr/spool/ppd/status3812
```

Error messages are recorded in the log file (*lf* in */etc/printcap*). The following command monitors the 3812 error log while documents are printing:

```
tail -f /usr/adm/ppd-errs
```

The error log (*lf*) and accounting file (*af*) will grow as documents are printed, and should be truncated periodically by your system administrator.

The print server (*ibm3812pp*(8)) runs as a daemon. It can be terminated by issuing the *kill*(1) command from root.

3. PRINTING TEXT FILES

Pprint(1) prints text files on the 3812. It filters input through *pr* and sends the resulting output to the spooling system. See the *pprint*(1) manual page for the various options. The file */etc/printcap* must contain an entry for the 3812 as described in Section 2.2.3, "Verifying Printcap Entries."

See Appendix A of this article for a list of fonts that can be specified with the **-f** option of *pprint*.

4. PRINTING DITROFF FILES

With the ditroff feature installed, device independent files are generated by */usr/ibm/troff*. The new command *ptroff*(1) processes *troff* source for printing on the 3812. See the *ptroff*(1) manual page for the various options. The file */etc/printcap* must contain an entry for the 3812 as described in Section 2.2.3, "Verifying Printcap Entries." See Appendix A of this article for a list of fonts that are available for use with *troff*, and Appendix B for a list of *troff* character names.

Ditroff includes modifications to *troff*, *eqn*, *pic*, and a *makedev* in Documenter's Workbench. These routines have been modified to use the 3812 fonts (**-T3812**); otherwise, the documentation for *troff*, *pic*, and *eqn* has not changed. The output file generated by *troff* has not been redefined. Only the format of the width tables has been changed. The width tables in Documenter's Workbench contain widths for only one character size, size 12. The width of a character in other sizes is calculated by scaling the base width: if the A in size 12 is 32 pels, the A in size 6 is 16 pels, and the A in size 24 is 64 pels. The 3812 fonts are designed individually for each size, so the width tables have been augmented to contain the width of every character in every size.

Because of the changes to the width tables, use */usr/ibm/troff*, */usr/ibm/pic*, and */usr/ibm/eqn* with **-T3812**. You can expect errors if you use */usr/bin/troff* or */usr/bin/eqn* with **-T3812**.

5. FONTS

The 3812 uses either the fonts provided on the 3812 diskette or fonts that are downloaded from the host. There are two sets of fonts: a set of uniformly-spaced fonts is provided with 4.2/RT for downloading, and several typographical fonts can be ordered from IBM as separate products. This chapter describes both groups of fonts.

5.1. Typographic Fonts

5.1.1. Ordering Typographic Fonts

The available typographic fonts include:

Sonoran Serif[10]	Program Number 5669-161, Feature 5124
Sonoran Sans Serif[11]	Program Number 5669-162, Feature 5125
Pi & Special[12]	Program Number 5669-163, Feature 5126

These fonts are shipped on diskettes. Samples of each font are shown in the *IBM 3800 Printing Subsystem Model III Font Catalog*, SH35-0053, and in Appendix A of this article.

5.1.2. Installing Typographic Fonts

When the typographic fonts are installed in */usr/lib/font/dev3812/fonts*, they are available for use with *pprint*(1) for printing on the 3812. To use these fonts with ditroff, the ditroff feature for the 3812 must be ordered and installed according to its installation procedures. Licensed ditroff users should refer to the *Program Directory - Ditroff for the IBM 3812 Pageprinter* for installation information.

(1) Login as root.

(2) Load the fonts from diskette.

There are three subdirectories in */usr/src/usr.lib/font/dev3812/fonts* that contain makefiles and tables for building fonts. Use *dosread*(1) to load the fonts from diskette to the IBM RT PC into the appropriate subdirectory.

(3) To load the Sonoran Serif font (Program Number 5669-161, Feature 5124), follow these steps:

```
cd /usr/src/usr.lib/font/dev3812/fonts/serif
```

Insert the first diskette of the Sonoran Serif font into the diskette reader, then type the following:

```
dosread
```

Repeat this step for all 12 diskettes in the Sonoran Serif font.

To load the Sonoran Sans Serif font (Program Number 5669-162, Feature 5125), follow these steps:

```
cd /usr/src/usr.lib/font/dev3812/fonts/sans
```

Insert the first diskette of the Sonoran Sans Serif font into the diskette reader, then type the following:

[10] Functional equivalent of Monotype Times New Roman, a trademark of The Monotype Corporation, Limited. Contains data derived under license from The Monotype Corporation, Limited.

[11] Functional equivalent of Monotype Ariel, a trademark of The Monotype Corporation, Limited. Contains data derived under license from The Monotype Corporation, Limited.

[12] Contains data derived under license from The Monotype Corporation, Limited.

```
dosread
```

Repeat this step for all 12 diskettes in the Sonoran Sans Serif font.

To load the Pi & Special font (Program Number 5669-163, Feature 5126), follow these steps:

```
cd /usr/src/usr.lib/font/dev3812/fonts/pispecial
```

Insert the first diskette of the Pi & Special font into the diskette reader, then type the following:

```
dosread
```

Repeat this step for both of the diskettes in the Pi & Special font.

Loading all the fonts from diskette takes approximately twenty-five minutes.

(4) Convert the fonts to 3812 format.

Makefile in */usr/src/usr.lib/font/dev3812/fonts* descends into the subdirectories and performs the makes on them.

Make automatically converts the fonts into the 3812 format (*cvt3812*(8)), and builds the width tables (*width3812*(8)). Type the following:

```
cd /usr/src/usr.lib/font/dev3812/fonts
make
```

This step takes approximately thirty minutes.

(5) Install the fonts for use with the 3812.

Make with the install option does the following:

- Copies the width tables to */usr/lib/font/dev3812.*
- Copies the font raster pattern to */usr/lib/font/dev3812/fonts.*

To install, type the following:

```
cd /usr/src/usr.lib/font/dev3812/fonts
make install
```

5.2. Uniformly-spaced Fonts

These fonts are provided in */usr/lib/font/dev3812/fonts.* They are shown in the *IBM 3812 Pageprinter Introduction and Planning Guide*, G544-3265, and the *IBM 3800 Printing Subsystem Model III Font Catalog*, SH35-0053. Their format is described in *font3812*(5).

10-pitch	12-pitch	Other fixed-pitch	Proportionally-spaced
APL	Courier *	13-pitch Letter Gothic	Barak
Courier *	Gothic Bold	15-pitch Gothic Text	Boldface *
Courier Italic	Gothic Italic	15-pitch Serif Text	Boldface Italic *
Gothic Bold	Gothic Text	15-pitch Shalom	Document *
Gothic Text	Letter Gothic *	20-pitch Shalom	Essay *
Katakana	Letter Gothic Bold *	20-pitch APL	Essay Bold *
Orator	Prestige Elite *	20-pitch APL	Essay Italic *
Orator Bold	Prestige Elite Bold *	20-pitch Gothic Text	Essay light *
Prestige Pica *	Prestige Elite Italic *	27-pitch Gothic Text	Gothic Tri-pitch
Roman Text	Script	15-pitch Format	
Serif Italic	Serif Bold		
Serif Text	Serif Italic		
Shalom	Shalom		
Math Symbol	Math Symbol		
OCR-A **	Format		
OCR-B **			
Format			

* These fonts contain international character sets of 221 printable characters.

** The 3812 prints the OCR-A and OCR-B fonts with the same high quality as other type styles. IBM does not warrant and has not tested that these characters are readable by all OCR reading devices. Users of these fonts should test read compatibility before relying on the 3812 for OCR applications.

6. USING CODE PAGE TABLES

A code page table identifies the set of characters to be used from one or more fonts. A font contains the graphic pel pattern for each character along with its IBM character name. An entry in a code page table associates an IBM character name with a *troff* character name. The entry's position in the table gives the character's ASCII code.

Lowercase "a", for example, has the IBM character name LA010000 regardless of font or point size, and would be represented by a code page table entry containing "a LA010000". The ASCII code is 97, so the entry is the 97th unique table entry.

Similarly, the pound symbol (£), if it is in the table at all, is represented by an entry containing "£ SC020000". Since it has no ASCII code, the entry may appear in any unused position.

A code page table may contain up to 256 unique entries, plus synonym entries. Appendix A of this article gives a list of the fonts and their corresponding code page tables; Appendix B lists the information appearing in the code page tables.

Two examples are given here. The first example shows how to modify an existing code page table for use with ditroff. The second shows how to make a new code page table.

Example 1

If you want to change the troff name assigned to a character, or if you want to change the graphic associated with a troff name, you must modify the code page table accordingly. For example, to change the troff names of left double quote \(LQ and right double quote \(RQ to \(L" and \(R" respectively, you would change the troff names in code page table *stdcp*. Then you would need to rebuild the width tables for all fonts that use that code page table and copy the resulting width tables to */usr/lib/font/dev3812*. Copy the resulting code page index files to */usr/lib/font/dev3812/fonts*.

In this example, R, I, B, BI, H, HI, HB, HY, D, and SP use the *stdcp* code page table and need to be rebuilt. You would rebuild the width tables by typing the following:

```
cd /usr/src/usr.lib/font/dev3812/fonts/serif
make serif
cp R I B BI /usr/lib/font/dev3812
cp *.stdcp /usr/lib/font/dev3812/fonts

cd /usr/src/usr.lib/font/dev3812/fonts/sans
make sans
cp H HI HB HY /usr/lib/font/dev3812
cp *.stdcp /usr/lib/font/dev3812/fonts

cd /usr/src/usr.lib/font/dev3812/fonts/pi
make special
cp D SP /usr/lib/font/dev3812
cp *.stdcp /usr/lib/font/dev3812/fonts
```

Next, build the binary form of the widths. This step depends on having the ditroff feature installed. Go to */usr/lib/font/dev3812* and build the binary form of the troff width tables:

```
cd /usr/lib/font/dev3812
makedev R I B BI H HI HB HY D SP DESC
```

Now the new character name is ready to be used by *troff*.

Example 2

The APL font is built using the *fcp* code page table, but there are IBM character names in the APL font that are not referenced by *.fcp.* To determine all the IBM character names in a font use the -N option on *cvt3812*(8). The following command generates all the IBM names for characters in the APL font:

```
cvt20to12 -N c0s0ae10 > /tmp/names
```

The IBM character names are documented in the *IBM 3800 Printing Subsystem Model III Font Catalog*, SH35-0053. Each element of */tmp/names* has the following format:

```
raster for x xx SL110000
```

SL11000 is the IBM character name. The "xx" in the list of character names is replaced with a *troff* character name. For example, the following will assign the name \(CS to the APL character Circle Star:

```
raster for x CS SL110000
```

Copy *fcp* to *aplcp* and add the new entries from */tmp/names* to *aplcp* into unused positions. Once the characters have been assigned a name, use *width3812*(8) to build the width table and the code page index files:

```
width3812 -s 5 10 0 -c aplcp -n AP APL
```

This will build the AP width tables for sizes 5 and 10, using the code page table *aplcp.* The width table (*AP*) must be copied to */usr/lib/font/dev3812* to be used by *troff.* The code page index files (*APL.10.aplcp* and *APL.5.aplcp*) must be copied to */usr/lib/font/dev3812/fonts.* This file can now be used with *pprint*(1). The following command will print a file using the APL font.

```
pprint -fAPL.10.aplcp filename
```

To use the font in a *troff* document, use *makedev* to generate the binary form of the troff width table:

```
cd /usr/lib/font/dev3812
makedev AP
```

Now use the *troff* commands for changing fonts (.ft AP) to print characters from the font.

APPENDIX A. FONTS AVAILABLE ON THE 3812

This appendix shows examples of the fonts available on the 3812. The typographic fonts are available for license from IBM; the uniformly-spaced fonts are provided with 4.2/RT.

Typographic Fonts

The following fonts are available for license from IBM. All are in the Sonoran type face, except for the Display and Petite fonts. Unless otherwise indicated, all are in point sizes 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 30, 36. Samples of these fonts are found later in this section.

Sonoran Serif Typeface

ptroff Name	*pprint* Name	Point Size	Code Page Table	Notes	Font Name
R	s	All	stdcp	(1)	Serif Roman
I	s.I	All	stdcp	(1)	Serif Italic
B	s.B	All	stdcp	(1)	Serif Bold
BI	s.BI	All	stdcp	(1)	Serif Bold Italic
S	-	6-12	picp	(2)	Serif PI (symbols)
SB	-	6-12	picp	(2)	Serif PI Bold (symbols)
L	-	All	latincp		Serif Roman Latin Characters
LI	-	All	latincp		Serif Italic Latin Characters
LB	-	All	latincp		Serif Bold Latin Characters
LY	-	All	latincp		Serif Bold Italic Latin Characters
A	-	6-12	addcp	(2)	Serif Additional Characters

Sonoran Sans Serif Typeface

ptroff Name	*pprint* Name	Point Size	Code Page Table	Notes	Font Name
H	ss	All	stdcp	(1)	Sans Serif Roman
HI	ss.I	All	stdcp	(1)	Sans Serif Italic
HB	ss.B	All	stdcp	(1)	Sans Serif Bold
HY	ss.BI	All	stdcp	(1)	Sans Serif Bold Italic
HS	-	6-12	picp	(2)	Sans Serif PI (symbols)
HZ	-	6-12	picp	(2)	Sans Serif PI Bold (symbols)
K	-	All	latincp		Sans Serif Roman Latin Characters
KI	-	All	latincp		Sans Serif Italic Latin Characters
KB	-	All	latincp		Sans Serif Bold Latin Characters
KY	-	All	latincp		Sans Serif Bold Italic Latin Characters

Notes:

(1) When using *pprint*, these fonts lack the ~ and ^ characters.

(2) Font sizes 7, 9, and 11 are duplicates of 6, 8, and 10 respectively.

Special Fonts

ptroff Name	*pprint* Name	Point Size	Code Page-Table	Notes	Font Name
D	d	20, 36	stdcp	(1)	Display
SP	pe	4	stdcp	(2)	Petite

Uniformly-Spaced Fonts

The following fonts are provided for use with *pprint*(1) and *ptroff*(1). These fonts are built with either the *fcp* or the *acp* code page table and contain all the ASCII characters. Samples of these fonts are found in the *IBM 3800 Printing Subsystem Model III Font Catalog*, SH35-0053.

ptroff Name	*pprint* Name	Point Size	Code Page Table	Notes	Font Name
Bb	BOOK.B	10	fcp		Book Bold (proportionally spaced)
Bi	BOOK.BI	10	fcp		Book Bold Italic (proportionally spaced)
CW	COURIER	10	fcp		Courier
Cw	Courier	4, 10	acp	(3)	Courier
Cb	Courier.B	10	acp	(3)	Courier Bold
Cc	Courier.C	10	acp	(3)	Courier Condensed
Cd	Courier.CB	10	acp	(3)	Courier Condensed Bold
Ce	Courier.E	10	acp	(3)	Courier Expanded
Cf	Courier.EB	10	acp	(3)	Courier Expanded Bold
Du	DOCUMENT	10	fcp		Document (proportionally spaced)
E	ESSAY	10	fcp		Essay (proportionally spaced)
EB	ESSAY.B	10	fcp		Essay Bold (proportionally spaced)
EI	ESSAY.I	10	fcp		Essay Italic (proportionally spaced)
EL	ESSAY.L	10	fcp		Essay Light (proportionally spaced)
Lr	LETTER	9	fcp		Letter Gothic
Lb	LETTER.B	9	fcp		Letter Gothic Bold
PP	PRESTIGE	9, 10	fcp		Prestige
PB	PRESTIGE.B	9	fcp		Prestige Bold
PI	PRESTIGE.I	9	fcp		Prestige Italic

Notes:

(1) When using *pprint*, the Display font lacks the following characters:

+ < = > @ [ ] ^ _ { } ~

(2) When using *pprint*, the Petite font lacks the ~ and ^ characters.

(3) These fonts are IBM 5152 Printer Emulation fonts.

Examples of Typographic Fonts

The Sonoran fonts shown here are printed in 10-point sizes, with a vertical spacing of 12 points and with non-alphanumeric characters separated by 1/4-em space. These examples are representative selections of the available characters in each font; not every available character is shown. See Appendix B for the code page tables for each font.

Serif Fonts

Serif Roman (R)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
• – - — ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢
” ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Serif Bold (B)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
• – - — ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢
” ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Serif Italic (I)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
• – - — ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢
” ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Serif Bold Italic (BI)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
• – - — ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢
” ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Serif PI - symbols (S)

α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π
ρ σ τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ‾ ≥ ≤ ≡ ∼ ≅ ≠ → ← ↑ ↓
× ÷ ± ∪ ∩ ⊂ ⊃ ∞ ∂ ® ©
^ ˜ − = ∫ ∝ ∅ ∈ ☎ ○ □

Serif PI Bold - symbols (SB)

α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π
ρ σ τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ‾ ≥ ≤ ≡ ∼ ≅ ≠ → ← ↑ ↓
× ÷ ± ∪ ∩ ⊂ ⊃ ∞ ∂ ® ©
^ ˜ − = ∫ ∝ ∅ ∈ ☎ ○ □

Serif Additional Characters (A)

⎡ ⎣ ⎤ ⎦ ⎨ ⎬ ⎪ ⎪ ⌊ ⌋ ⌈ ⌉ ς ∇ ⊆ ⊇

Sans Serif Fonts

Sans Serif Roman (H)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |
• — - — ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢
" ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Sans Serif Bold (HB)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |
• — - — ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢
" ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Sans Serif Italic (HI)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |
• — - — ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢
" ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Sans Serif Bold Italic (HY)

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |
• — - — ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢
" ´ \ _ ` / < > { } # @ + * § ¬ ‡
» «

Sans Serif PI - symbols (HS)

α β γ δ ϵ ζ η θ ι κ λ μ ν ξ ο π ρ σ
τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ‾ ≥ ≤ ≡ ~ ≅ ≠ → ← ↑ ↓
× ÷ ± ∪ ∩ ⊂ ⊃ ∞ ∂ ® ©
^ ˜ – = ∫ ∝ ∅ ∈ ☎ ○ □

Sans Serif PI Bold - symbols (HZ)

α β γ δ ϵ ζ η θ ι κ λ μ ν ξ ο π ρ
σ τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ‾ ≥ ≤ ≡ ~ ≅ ≠ → ← ↑ ↓
× ÷ ± ∪ ∩ ⊂ ⊃ ∞ ∂ ® ©
^ ˜ – = ∫ ∝ ∅ ∈ ☎ ○ □

Special Fonts

Two special fonts are available. Both are built with code page table *stdcp*.

The Display font is printed in 20 points, with a vertical spacing of 30 points and with non-alphanumeric characters separated by 1/4-em space. This font is available only in point sizes 20 and 36.

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * - " " . , / : ; ?

The Petite font is printed in 4 points, with a vertical spacing of 6 points and with non-alphanumeric characters separated by 1/4-em space. This font is available only in a 4-point size.

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + - . , / : ; = ? [ ] |

Latin Characters

The Latin and special characters shown below are available in both the Sonoran Serif and Sans Serif font families in roman, italic, bold, and bold italic. They are built with code page table *latincp*.

á Á à À â Â ä Ä Ã å Å æ Æ ç Ç é É è È ê Ê ë
Ë ĕ Ĕ í Í ì Ì î Î ï Ï ĩ Ĩ ij ł Ŀ ń Ń ñ Ñ ň Ň
ó Ó ò Ò ô Ô ö Ö õ Õ œ Œ ø Ø ß ú Ú ù Ù û Û ü
Ü ũ Ũ ů Ů ÿ Ÿ ± < = > ÷ × ≤ ≥ ≠ ¤ ¥ ₧ § ¶
‰ ¡ ′ ( ) , - · ¿ „ 🄽

APPENDIX B. CODE PAGE TABLES

This appendix lists the characters in each code page, with their *troff*, IBM, and descriptive names. Refer to files in */usr/src/usr.lib/font/dev3812/fonts/*cp* for precise code page contents.

STANDARD CODE PAGE (stdcp)

Char	Troff Char Name	IBM Char Name	Description
ﬀ	\(ff	LF510000	ff ligature
ﬁ	\(fi	LF530000	fi ligature
ﬂ	\(fl	LF550000	fl ligature
ﬃ	\(Fi	LF570000	ffi ligature
ﬄ	\(Fl	LF590000	ffl ligature
|	\(br	SM130000	Vertical bar, logical OR
½	\(12	NF010000	Numeric fraction one-half
¼	\(14	NF040000	Numeric fraction one-quarter
¾	\(34	NF050000	Numeric fraction three-quarters
⅛	\(18	NF180000	Numeric fraction one-eighth
⅜	\(38	NF190000	Numeric fraction three-eighths
⅝	\(58	NF200000	Numeric fraction five-eighths
⅞	\(78	NF210000	Numeric fraction seven-eighths
†	\(dg	SM340000	Dagger
‡	\(dd	SM350000	Double dagger
¦	\(VB	SM650000	Vertical broken line
°	\(de	SM190000	Degree symbol
f	\(fg	SC070000	Florin or guilder
|	\(or	SM130000	Vertical bar, logical OR
′	\(fm	SP050000	Apostrophe
•	\(bu	SM570000	Bullet
§	\(sc	SM240000	Section symbol (USA), paragraph symbol (Europe)
-	\(hy	SP100000	Hyphen
“	\(LQ	SP210000	Left double quotes
”	\(RQ	SP220000	Right double quotes
¢	\(ct	SC040000	Cent
£	\(Lb	SC020000	Pound
−	\(mi	SA000000	Minus
¬	\(no	SM660000	Logical NOT
−	\(em	SA000000	Minus
″	\(QM	SP040000	Quotation marks
−	\-	SA000000	Minus
	\(SP	SP010000	Interword space
!	!	SP020000	Exclamation point
”	”	SP220000	Right double quotes
#	#	SM010000	Number
$	$	SC030000	Dollar
%	%	SM020000	Percent
&	&	SM030000	Ampersand
(	(	SP060000	Left parenthesis
)	)	SP070000	Right parenthesis
*	*	SM040000	Asterisk
*	\(**	SM040000	Asterisk

+	+	SA010000	Plus
,	,	SP080000	Comma
-	-	SP100000	Hyphen
/	/	SP120000	Slash
0	0	ND100000	Numeric decimal zero
1	1	ND010000	Numeric decimal one
2	2	ND020000	Numeric decimal two
3	3	ND030000	Numeric decimal three
4	4	ND040000	Numeric decimal four
5	5	ND050000	Numeric decimal five
6	6	ND060000	Numeric decimal six
7	7	ND070000	Numeric decimal seven
8	8	ND080000	Numeric decimal eight
9	9	ND090000	Numeric decimal nine
:	:	SP130000	Colon
;	;	SP140000	Semicolon
<	<	SA030000	Less-than
=	=	SA040000	Equals
>	>	SA050000	Greater-than
?	?	SP150000	Question mark
@	@	SM050000	At
A	A	LA020000	A capital
B	B	LB020000	B capital
C	C	LC020000	C capital
D	D	LD020000	D capital
E	E	LE020000	E capital
F	F	LF020000	F capital
G	G	LG020000	G capital
H	H	LH020000	H capital
I	I	LI020000	I capital
J	J	LJ020000	J capital
K	K	LK020000	K capital
L	L	LL020000	L capital
M	M	LM020000	M capital
N	N	LN020000	N capital
O	O	LO020000	O capital
P	P	LP020000	P capital
Q	Q	LQ020000	Q capital
R	R	LR020000	R capital
S	S	LS020000	S capital
T	T	LT020000	T capital
U	U	LU020000	U capital
V	V	LV020000	V capital
W	W	LW020000	W capital
X	X	LX020000	X capital
Y	Y	LY020000	Y capital
Z	Z	LZ020000	Z capital
[	[	SM060000	Left bracket
\	\	SM070000	Reverse slash
]	]	SM080000	Right bracket
«	\(< <	SP170000	Left angle quotes
_	_	SP090000	Continuous underscore, underline
‘	‘	SP190000	Left single quote

a	a	LA010000	a small
b	b	LB010000	b small
c	c	LC010000	c small
d	d	LD010000	d small
e	e	LE010000	e small
f	f	LF010000	f small
g	g	LG010000	g small
h	h	LH010000	h small
i	i	LI010000	i small
j	j	LJ010000	j small
k	k	LK010000	k small
l	l	LL010000	l small
m	m	LM010000	m small
n	n	LN010000	n small
o	o	LO010000	o small
p	p	LP010000	p small
q	q	LQ010000	q small
r	r	LR010000	r small
s	s	LS010000	s small
t	t	LT010000	t small
u	u	LU010000	u small
v	v	LV010000	v small
w	w	LW010000	w small
x	x	LX010000	x small
y	y	LY010000	y small
z	z	LZ010000	z small
{	{	SM110000	Left brace
|	|	SM130000	Vertical bar, logical OR
}	}	SM140000	Right brace
»	\(> >	SP180000	Right angle quotes
▧	\(UN	SV320000	Replacement symbol (for undefined code points)
_	\(ul	SP090000	Continuous underscore, underline
—	\(ru	SM900000	Rule
«	\(lh	SP170000	Left angle quotes
»	\(rh	SP180000	Right angle quotes
+	\(pl	SA010000	Plus
/	\(sl	SP120000	Slash

LATIN CODE PAGE (latincp)

	Troff	IBM	
	Char	Char	
Char	Name	Name	Description
á	\(a'	LA110000	a acute small
Á	\(A'	LA120000	A acute capital
à	\(a‘	LA130000	a grave small
À	\(A‘	LA140000	A grave capital
â	\(a^	LA150000	a circumflex small
Â	\(A^	LA160000	A circumflex capital
ä	\(a:	LA170000	a diaeresis/umlaut small
Ä	\(A:	LA180000	A diaeresis/umlaut capital
Ã	\(A~	LA200000	A tilde capital
å	\(a.	LA270000	a overcircle small
Å	\(A.	LA280000	A overcircle capital
æ	\(ae	LA510000	ae diphthong small
Æ	\(AE	LA520000	AE diphthong capital
ç	\(c,	LC410000	c cedilla small
Ç	\(C,	LC420000	C cedilla capital
é	\(e'	LE110000	e acute small
É	\(E'	LE120000	E acute capital
è	\(e‘	LE130000	e grave small
È	\(E‘	LE140000	E grave capital
ê	\(e^	LE150000	e circumflex small
Ê	\(E^	LE160000	E circumflex capital
ë	\(e:	LE170000	e diaeresis/umlaut small
Ë	\(E:	LE180000	E diaeresis/umlaut capital
ĕ	\(e-	LE210000	C caron small
Ĕ	\(E-	LE220000	E caron capital
í	\(i'	LI110000	i acute small
Í	\(I'	LI120000	I acute capital
ì	\(i‘	LI130000	i grave small
Ì	\(I‘	LI140000	I grave capital
î	\(i^	LI150000	i circumflex small
Î	\(I^	LI160000	I circumflex capital
ï	\(i:	LI170000	i diaeresis/umlaut small
Ï	\(I:	LI180000	I diaeresis/umlaut capital
ĩ	\(i~	LI190000	i tilde small
Ĩ	\(I~	LI200000	I tilde capital
ij	\(ij	LI510000	ij ligature small
ŀ	\(l.	LL630000	l middle dot small
Ŀ	\(L.	LL640000	L middle dot capital
ń	\(n'	LN110000	n acute small
Ń	\(N'	LN120000	N acute capital
ñ	\(n~	LN190000	n tilde small
Ñ	\(N~	LN200000	N tilde capital
ň	\(n-	LN210000	n caron small
Ň	\(N-	LN220000	N caron capital
ó	\(o'	LO110000	o acute small
Ó	\(O'	LO120000	O acute capital
ò	\(o‘	LO130000	O grave small

Ò	\(O‘	LO140000	O grave capital
ô	\(o^	LO150000	o circumflex small
Ô	\(O^	LO160000	O circumflex capital
ö	\(o:	LO170000	o diaeresis/umlaut small
Ö	\(O:	LO180000	O diaeresis/umlaut capital
õ	\(o˜	LO190000	o tilde small
Õ	\(O˜	LO200000	O tilde capital
œ	\(oe	LO510000	oe diphthong small
Œ	\(OE	LO520000	OE diphthong capital
ø	\(o/	LO610000	o slash small
Ø	\(O/	LO620000	O slash capital
ß	\(ss	LS610000	s sharp small
ú	\(u’	LU110000	u acute small
Ú	\(U’	LU120000	U acute capital
ù	\(u‘	LU130000	u grave small
Ù	\(U‘	LU140000	U grave capital
û	\(u^	LU150000	u circumflex small
Û	\(U^	LU160000	U circumflex capital
ü	\(u:	LU170000	u diaeresis/umlaut small
Ü	\(U:	LU180000	U diaeresis/umlaut capital
ũ	\(u˜	LU190000	u tilde small
Ũ	\(U˜	LU200000	U tilde capital
ů	\(u.	LU270000	u overcircle small
Ů	\(U.	LU280000	U overcircle capital
ÿ	\(y:	LY170000	y diaeresis/umlaut small
Ÿ	\(Y:	LY180000	Y diaeresis/umlaut capital
±	\(S-	SA021000	Plus-or-minus superscript
<	\(S<	SA031000	Less-than superscript
=	\(S=	SA041000	Equals superscript
>	\(S>	SA051000	Greater-than superscript
÷	\(S/	SA061000	Divide superscript
×	\(Sx	SA071000	Multiply superscript
≤	\(Sl	SA521000	Less-than-or-equal superscript
≥	\(Sg	SA531000	Greater-than-or-equal superscript
≠	\(S!	SA541000	Not-equal superscript
¤	\(Ic	SC010000	International currency
¥	\(Y-	SC050000	Yen
₧	\(Ps	SC060000	Peseta
§	\(SS	SM240000	Section symbol (USA), paragraph symbol (Europe)
¶	\(PS	SM250000	Paragraph symbol (USA)
‰	\(PM	SM560000	Per mill symbol
¡	\(I!	SP030000	Exclamation point, inverted
′	\(R’	SP050000	Apostrophe
(	\(So	SP061000	Left parenthesis superscript
)	\(Sc	SP071000	Right parenthesis superscript
,	\(Sm	SP081000	Comma superscript
-	\(Sh	SP101000	Hyphen superscript
·	\(Sp	SP111000	Period, full stop superscript
¿	\(I?	SP160000	Question mark, inverted
„	\(L:	SP230000	Left lower double quotes (German)
	\(RS	SP300000	Required space
⍰	\(UN	SV320000	Replacement symbol (for undefined code points)

PI CODE PAGE (picp)

Char	Troff Char Name	IBM Char Name	Description
▧	\(^@	SV320000	Replacement symbol (for undefined codepoints)
α	\(*a	GA010000	Alpha small
β	\(*b	GB010000	Beta small
δ	\(*d	GD010000	Delta small
ε	\(*e	GE010000	Epsilon small
η	\(*y	GE310000	Eta small
φ	\(*f	GF010000	Phi small
γ	\(*g	GG010000	Gamma small
χ	\(*x	GH010000	Chi small
ι	\(*i	GI010000	Iota small
κ	\(*k	GK010000	Kappa small
λ	\(*l	GL010000	Lambda small
μ	\(*m	GM010000	Mu small
ν	\(*n	GN010000	Nu small
ο	\(*o	GO010000	Omicron small
ω	\(*w	GO310000	Omega small
π	\(*p	GP010000	Pi small
ψ	\(*q	GP610000	Psi small
ρ	\(*r	GR010000	Rho small
σ	\(*s	GS010000	Sigma small
τ	\(*t	GT010000	Tau small
θ	\(*h	GT610000	Theta small
υ	\(*u	GU010000	Upsilon small
ξ	\(*c	GX010000	Xi small
ζ	\(*z	GZ010000	Zeta small
Δ	\(*D	GD020000	Delta capital
Φ	\(*F	GF020000	Phi capital
Γ	\(*G	GG020000	Gamma capital
Λ	\(*L	GL020000	Lambda capital
Ω	\(*W	GO320000	Omega capital
Π	\(*P	GP020000	Pi capital
Ψ	\(*Q	GP620000	Psi capital
Σ	\(*S	GS020000	Sigma capital
Θ	\(*II	GT620000	Theta small
Ξ	\(*C	GX020000	Xi capital
♂	\(Ma	SM280000	Male symbol
♀	\(Fe	SM290000	Female symbol
℞	\(Rx	SM550000	Prescription symbol
ℛ	\(Ri	SS470000	Riemann integral
ℒ	\(La	SS480000	LaPlace symbol
℘	\(Rn	SS460000	"Real number" symbol
ƒ	\(Fs	SS440000	Function symbol
↘	\(De	SM990000	Decrease
↗	\(In	SM950000	Increase
℁	\(AO	SS650000	"Account of" symbol
℅	\(CO	SS640000	"Care of"symbol
●	\(BS	SS630000	Bottle symbol

ɤ	\(PT	SS610000	"Plaintiff" symbol
★	\(CS	SS580000	Closed star upright
∵	\(TD	SS540000	"Because" symbol
∶	\(RA	SS430000	Ratio symbol
∑	\(SU	SS400000	Summation symbol
○	\(ci	SM750000	Open circle
△	\(OT	SM730000	Open triangle, mode change
◆	\(SD	SM610000	Solid diamond
▲	\(ST	SM600000	Solid triangle
▶	\(AI	SM590000	Arrow indicator
™	\(TM	SM540000	Trademark symbol
®	\(rg	SM530000	Registered trademark symbol
©	\(co	SM520000	Copyright symbol
□	\(sq	SM450000	Open square
↓	\(da	SM330000	Down arrow
↑	\(ua	SM320000	Up arrow
→	\(->	SM310000	Right arrow
←	\(<-	SM300000	Left arrow
√	\(sr	SM230000	Tape mark, radical
	\(rn	SM150000	Overline
‗	\(DU	SM100000	Double underscore
¸	\(CE	SD410000	Cedilla
˜	~	SD190000	Tilde
¨	\(UM	SD170000	Diaeresis or umlaut
^	^	SD150000	Circumflex
`	\(ga	SD130000	Grave
´	\(aa	SD110000	Acute
·	\(Md	SA790000	Dot multiply, middle dot
⊥	\(Pd	SA780000	Perpendicular to
≈	\(Ae	SA700000	Nearly equals
∊	\(mo	SA670000	"Is an element of"
◇	\(Di	SA660000	Diamond
⊕	\(Cs	SA550000	Closed sum
∫	\(is	SA510000	Integral symbol
∂	\(pd	SA490000	Partial differential symbol
≡	\(==	SA480000	Identity symbol
∝	\(pt	SA470000	Proportional to
∞	\(if	SA450000	Infinity symbol
≅	\(~=	SA430000	Congruent to
⊃	\(sp	SA410000	Includes, a superset of
⊂	\(sb	SA400000	Included in, a subset of
∪	\(cu	SA390000	Union, logical sum
∩	\(ca	SA380000	Intersection, logical product
∴	\(Tf	SA370000	Therefore symbol
∉	\(Nm	SA360000	Is not an element of
∠	\(An	SA350000	Angle symbol
∥	\(Pl	SA340000	Parallel symbol
∼	\(ap	SA160000	Cycle symbol, equivalent to
ý	\(Y1	LY110000	y acute small
Ý	\(Y2	LY120000	Y acute capital
þ	\(T1	LT630000	Thorn Icelandic small
Þ	\(T2	LT640000	Thorn Icelandic capital
ł	\(L1	LL610000	l stroke small

Ł	\(L2	LL620000	L stroke capital
ı	\(I1	LI610000	i dotless small
ę	\(E1	LE430000	e ogonek small
Ę	\(E2	LE440000	E ogonek capital
ð	\(D3	LD630000	eth Icelandic small
đ	\(D1	LD610000	d stroke small
Đ	\(D2	LD620000	D stroke capital and Eth Icelandic capital
ą	\(U1	LA430000	a ogonek small
Ą	\(U2	LA440000	A ogonek capital
⑀	\(HO	SO000000	Hook
⑂	\(FO	SO010000	Fork
⑁	\(CH	SO020000	Chair
☏	\(TO	SS670000	Telephone symbol (open)
☎	\(TC	SS700000	Telephone symbol (closed)
∅	\(0/	ND100008	Numeric decimal zero slash
ℓ	\(Lt	SM160000	Litre symbol
º	\(ML	SM200000	Ordinal indicator - masculine
ª	\(FE	SM210000	Ordinal indicator - feminine
ƀ	\(b/	SM670000	Substitute blank
–	\(en	SS680000	En-dash
−	\(mi	SA000000	Minus
+	+	SA010000	Plus
±	\(+-	SA020000	Plus-or-minus
=	\(eq	SA040000	Equals
÷	\(di	SA060000	Divide
×	\(mu	SA070000	Multiply
≤	\(<=	SA520000	Less-than-or-equal
≥	\(>=	SA530000	Greater-than-or-equal
≠	\(!=	SA540000	Not-equal
¦	\(BV	SM650000	Vertical broken line
Α	\(*A	GA020000	Alpha capital
Β	\(*B	GB020000	Beta capital
Ε	\(*E	GE020000	Epsilon capital
Ι	\(*I	GI020000	Iota capital
Κ	\(*K	GK020000	Kappa capital
Μ	\(*M	GM020000	Mu capital
Ν	\(*N	GN020000	Nu capital
Ο	\(*O	GO020000	Omicron capital
Ρ	\(*R	GR020000	Rho capital
Τ	\(*T	GT020000	Tau capital
Υ	\(*U	GU020000	Upsilon capital
Χ	\(*X	GH020000	Chi capital
Η	\(*Y	GE320000	Eta capital
Ζ	\(*Z	GZ020000	Zeta capital
^{0}	\(0S	ND101000	Numeric decimal zero superscript
^{1}	\(1S	ND011000	Numeric decimal one superscript
^{2}	\(2S	ND021000	Numeric decimal two superscript
^{3}	\(3S	ND031000	Numeric decimal three superscript
^{4}	\(4S	ND041000	Numeric decimal four superscript
^{5}	\(5S	ND051000	Numeric decimal five superscript
^{6}	\(6S	ND061000	Numeric decimal six superscript
^{7}	\(7S	ND071000	Numeric decimal seven superscript
^{8}	\(8S	ND081000	Numeric decimal eight superscript

^{9}	\(9S	ND091000	Numeric decimal nine superscript
	\(IW	SP010000	Interword space
⌑	\(LZ	SM490000	Lozenge
′	\(MI	SM500000	Minutes symbol
¬	\(NO	SM660000	Logical NOT, end of line
$^{(}$	\(So	SP061000	Left parenthesis superscript
$^{)}$	\(Sc	SP071000	Right parenthesis superscript
,	\(Sm	SP081000	Comma superscript
$^{.}$	\(Sp	SP111000	Period, full stop superscript
″	\(SE	SM510000	Seconds symbol
■	\(Ss	SM470000	Solid square, histogram
☎	\(bs	SS700000	Telephone symbol, closed
^{a}	\(aS	LA011000	a small superscript
^{b}	\(bS	LB011000	b small superscript
^{c}	\(cS	LC011000	c small superscript
^{d}	\(dS	LD011000	d small superscript
^{e}	\(eS	LE011000	e small superscript
^{f}	\(fS	LF011000	f small superscript
^{g}	\(gS	LG011000	g small superscript
^{h}	\(hS	LH011000	h small superscript
^{i}	\(iS	LI011000	i small superscript
^{j}	\(jS	LJ011000	j small superscript
^{k}	\(kS	LK011000	k small superscript
^{l}	\(lS	LL011000	l small superscript
^{m}	\(mS	LM011000	m small superscript
^{n}	\(nS	LN011000	n small superscript
^{o}	\(oS	LO011000	o small superscript
^{p}	\(pS	LP011000	p small superscript
^{q}	\(qS	LQ011000	q small superscript
^{r}	\(rS	LR011000	r small superscript
^{s}	\(sS	LS011000	s small superscript
^{t}	\(tS	LT011000	t small superscript
^{u}	\(uS	LU011000	u small superscript
^{v}	\(vS	LV011000	v small superscript
^{w}	\(wS	LW011000	w small superscript
^{x}	\(xS	LX011000	x small superscript
^{y}	\(yS	LY011000	y small superscript
^{z}	\(zS	LZ011000	z small superscript

ADDITIONAL CODE PAGE (addcp)

Char	Troff Char Name	IBM Char Name	Description
⎧	\(lt	ACIS0001	left top of big curly bracket
⎨	\(lk	ACIS0002	left center of big curly bracket
⎩	\(lb	ACIS0003	left bottom of big curly bracket
|	\(bv	ACIS0004	bold vertical
⎫	\(rt	ACIS0005	right top of big curly bracket
⎬	\(rk	ACIS0006	right center of big curly bracket
⎭	\(rb	ACIS0007	right bottom of big curly bracket
⊆	\(ib	ACIS0008	improper subset
⊇	\(ip	ACIS0009	improper superset
∇	\(gr	ACIS0010	gradient
⌊	\(lf	ACIS0011	left floor (left bottom of big square bracket)
⌈	\(lc	ACIS0012	left ceiling (top left)
⌋	\(rf	ACIS0013	right floor (right bottom)
⌉	\(rc	ACIS0014	right ceiling (right top)
ς	\(ts	ACIS0015	terminal sigma

FIXED CODE PAGE (fcp and acp)

Char	Troff Char Name	IBM Char Name	Description
^{0}	\(0S	ND101000	Numeric decimal zero superscript
^{1}	\(1S	ND011000	Numeric decimal one superscript
^{2}	\(2S	ND021000	Numeric decimal two superscript
^{3}	\(3S	ND031000	Numeric decimal three superscript
^{4}	\(4S	ND041000	Numeric decimal four superscript
^{5}	\(5S	ND051000	Numeric decimal five superscript
^{6}	\(6S	ND061000	Numeric decimal six superscript
^{7}	\(7S	ND071000	Numeric decimal seven superscript
^{8}	\(8S	ND081000	Numeric decimal eight superscript
^{9}	\(9S	ND091000	Numeric decimal nine superscript
¾	\(34	NF050000	Numeric fraction three-quarters
₢	\(Cx	SC090000	Cruzeiro
ª	\(FE	SM210000	Ordinal indicator - feminine
º	\(ML	SM200000	Ordinal indicator - masculine
¶	\(PS	SM250000	Paragraph symbol (USA)
	\(RS	SP300000	Required space
§	\(SS	SM240000	Section symbol (USA), paragraph symbol (Europe)
¨	\(UM	SD170000	Diaeresis or umlaut
$\boxed{N}$	\(UN	SV320000	Replacement symbol (for undefined code points)
'	\(fm	SP050000	Apostrophe
•	\(bu	SM570000	Bullet
ˋ	\(ga	SD130000	Grave
ĳ	\(ij	LI510000	ij ligature small
¡	\(I!	SP030000	Exclamation point, inverted
¤	\(Ic	SC010000	International currency
■	\(ss	LS610000	s sharp small
Á	\(A'	LA120000	A acute capital
Â	\(A^	LA160000	A circumflex capital
À	\(A‘	LA140000	A grave capital
Ã	\(A~	LA200000	A tilde capital
ã	\(a~	LA190000	a tilde small
	\(SP	SP010000	Interword space
!	!	SP020000	Exclamation point
"	”	SP220000	Right double quotes
#	#	SM010000	Number
$	$	SC030000	Dollar
%	%	SM020000	Percent
&	&	SM030000	Ampersand
(	(	SP060000	Left parenthesis
)	)	SP070000	Right parenthesis
*	*	SM040000	Asterisk
+	+	SA010000	Plus
,	,	SP080000	Comma
-	-	SP100000	Hyphen
/	/	SP120000	Slash
0	0	ND100000	Numeric decimal zero
1	1	ND010000	Numeric decimal one

2	2	ND020000	Numeric decimal two
3	3	ND030000	Numeric decimal three
4	4	ND040000	Numeric decimal four
5	5	ND050000	Numeric decimal five
6	6	ND060000	Numeric decimal six
7	7	ND070000	Numeric decimal seven
8	8	ND080000	Numeric decimal eight
9	9	ND090000	Numeric decimal nine
:	:	SP130000	Colon
;	;	SP140000	Semicolon
<	<	SA030000	Less-than
=	=	SA040000	Equals
>	>	SA050000	Greater-than
?	?	SP150000	Question mark
@	@	SM050000	At
A	A	LA020000	A capital
B	B	LB020000	B capital
C	C	LC020000	C capital
D	D	LD020000	D capital
E	E	LE020000	E capital
F	F	LF020000	F capital
G	G	LG020000	G capital
H	H	LH020000	H capital
I	I	LI020000	I capital
J	J	LJ020000	J capital
K	K	LK020000	K capital
L	L	LL020000	L capital
M	M	LM020000	M capital
N	N	LN020000	N capital
O	O	LO020000	O capital
P	P	LP020000	P capital
Q	Q	LQ020000	Q capital
R	R	LR020000	R capital
S	S	LS020000	S capital
T	T	LT020000	T capital
U	U	LU020000	U capital
V	V	LV020000	V capital
W	W	LW020000	W capital
X	X	LX020000	X capital
Y	Y	LY020000	Y capital
Z	Z	LZ020000	Z capital
[	[	SM060000	Left bracket
\	\	SM070000	Reverse slash
]	]	SM080000	Right bracket
^	^	SD150000	Circumflex
_	_	SP090000	Continuous underscore, underline
`	‘	SP190000	Left single quote
a	a	LA010000	a small
b	b	LB010000	b small
c	c	LC010000	c small
d	d	LD010000	d small
e	e	LE010000	e small
f	f	LF010000	f small

g	g	LG010000	g small
h	h	LH010000	h small
i	i	LI010000	i small
j	j	LJ010000	j small
k	k	LK010000	k small
l	l	LL010000	l small
m	m	LM010000	m small
n	n	LN010000	n small
o	o	LO010000	o small
p	p	LP010000	p small
q	q	LQ010000	q small
r	r	LR010000	r small
s	s	LS010000	s small
t	t	LT010000	t small
u	u	LU010000	u small
v	v	LV010000	v small
w	w	LW010000	w small
x	x	LX010000	x small
y	y	LY010000	y small
z	z	LZ010000	z small
{	{	SM110000	Left brace
¦	|	SM130000	Vertical bar, logical OR
}	}	SM140000	Right brace
˜	˜	SD190000	Tilde
»	\(> >	SP180000	Right angle quotes
🄽	\(UN	SV320000	Replacement symbol (for undefined code points)
Ç	\(C,	LC420000	C cedilla capital
ü	\(u:	LU170000	u diaeresis/umlaut small
é	\(e'	LE110000	e acute small
â	\(a^	LA150000	a circumflex small
ä	\(a:	LA170000	a diaeresis/umlaut small
à	\(a‘	LA130000	a grave small
å	\(a.	LA270000	a overcircle small
ç	\(c,	LC410000	c cedilla small
´	\(aa	SD110000	Acute (this is SP050000 in acp)
ë	\(e:	LE170000	e diaeresis/umlaut small
è	\(e‘	LE130000	e grave small
ï	\(i:	LI170000	i diaeresis/umlaut small
î	\(i^	LI150000	i circumflex small
ì	\(i‘	LI130000	i grave small
Ä	\(A:	LA180000	A diaeresis/umlaut capital
Å	\(A.	LA280000	A overcircle capital
É	\(E'	LE120000	E acute capital
æ	\(ae	LA510000	ae diphthong small
Æ	\(AE	LA520000	AE diphthong capital
ô	\(o^	LO150000	o circumflex small
ö	\(o:	LO170000	o diaeresis/umlaut small
ò	\(o‘	LO130000	O grave small
û	\(u^	LU150000	u circumflex small
ü	\(u:	LU170000	u diaeresis/umlaut small
ù	\(u‘	LU130000	u grave small
Ö	\(O:	LO180000	O diaeresis/umlaut capital
Ü	\(U:	LU180000	U diaeresis/umlaut capital

¢	\(ct	SC040000	Cent
£	\(Lb	SC020000	Pound
¥	\(Y-	SC050000	Yen
₧	\(Ps	SC060000	Peseta
ƒ	\(fg	SC070000	Florin or guilder
á	\(a'	LA110000	a acute small
í	\(i'	LI110000	i acute small
ó	\(o'	LO110000	o acute small
ú	\(u'	LU110000	u acute small
ñ	\(n~	LN190000	n tilde small
Ñ	\(N~	LN200000	N tilde capital
¿	\(I?	SP160000	Question mark, inverted
≠	\(!=	SA540000	Not-equal
¬	\(no	SM660000	Logical NOT, end of line
½	\(12	NF010000	Numeric fraction one-half
¼	\(14	NF040000	Numeric fraction one-quarter
Ë	\(E:	LE180000	E diaeresis/umlaut capital
Ê	\(E^	LE160000	E circumflex capital
È	\(E'	LE140000	E grave capital
Ï	\(I:	LI180000	I diaeresis/umlaut capital
Î	\(I^	LI160000	I circumflex capital
Ì	\(I'	LI140000	I grave capital
Ó	\(O'	LO120000	O acute capital
Ø	\(O/	LO620000	O slash capital
Ô	\(O^	LO160000	O circumflex capital
Ò	\(O'	LO140000	O grave capital
µ	\(MU	SM170000	Vertical bar, logical OR
Õ	\(O~	LO200000	O tilde capital
Ú	\(U'	LU120000	U acute capital
Ů	\(U.	LU280000	U overcircle capital
Û	\(U^	LU160000	U circumflex capital
Ù	\(U'	LU140000	U grave capital
ê	\(e^	LE150000	e circumflex small
ọ	\(o.	LO450000	o small underdot
±	\(+-	SA020000	Plus-or-minus
≥	\(>=	SA530000	Greater-than-or-equal
≤	\(<=	SA520000	Less-than-or-equal
÷	\(di	SA060000	Divide
°	\(de	SM190000	Degree symbol
−	-	SA000000	Minus
■	\(Ss	SM470000	Solid square, histogram

REFERENCES

You may want to obtain copies of the following documentation:

- *IBM 3812 Pageprinter Introduction and Planning Guide*, G544-3265
- *IBM 3812 Pageprinter Guide to Operations*, S544-3267
- *IBM 3812 Pageprinter Programming Reference*, S544-3268
- *IBM 3800 Printing Subsystem Model III Font Catalog*, SH35-0053

4.2/RT Linkage Convention

ABSTRACT

The 4.2/RT linkage convention provides an efficient method of calling, executing and returning from functions. The convention provides support for customary facilities of C, FORTRAN, and Pascal, including *varargs*, *alloca*, and profiling.

This article is intended for compiler writers and others who must write or analyze programs at the machine-instruction level. It presumes understanding of the RT PC architecture and the 4.2/RT assembler language.

Also described is the Floating Point Arithmetic linkage, which presents a low-overhead, uniform interface to the floating point hardware and its software emulation.

1. Introduction

A C function **foo** consists of a *text area* and a *data area*. The data area is named **_foo** and, in addition to quantities specified below, may contain constants and initialized variables. The text area contains machine instructions followed by a *trace table* that provides auxiliary information for debuggers.

Each call of **foo** creates a *stack frame* containing arguments, local variables, and space to save the caller's registers to be restored on return to the caller.

When **foo** is called, the caller first prepares an argument list, then transfers control to the text location named **_.foo**, which is **foo**'s entry point. **foo**'s *prolog* builds a stack frame to hold local variables and saves any registers that are to be preserved for the caller. Execution proceeds through the body of **foo**, possibly calling other functions, and ends in the *epilog*, which prepares the return value, restores the caller's registers, releases the stack frame, and transfers control back to the caller.

2. Stack Usage and Stack Frame Format

The stack holds frames for currently active functions and signal handlers. It is word-aligned and grows downward from approximately 0x1fffe000. A "red zone" of protected addresses separates the stack from the data segment, which starts at 0x10000000 and may grow upward as the result of *brk*(2) and *sbrk*(2) usage. Register r1 indicates the low address of the stack frame of the currently executing function. Locations above (r1) − 0x64 are preserved over interrupts. Locations below (r1) − 0x64 are considered unallocated storage and may be overwritten if a signal handler is activated.

The stack is not self-describing, but with information from the trace tables in program text, a debugger can decompose the stack into frames and backtrace through it.

foo's stack frame holds the following areas, from lowest address to highest:

a) Words 5 through *pmax* of outgoing argument lists. (*pmax* represents the number of words in the longest argument list for functions that **foo** calls.)

b) Local variables: autos and temporaries.

c) 0-8 words of save area for caller's floating point registers.

d) 1-16 words of save area for caller's general registers.

e) 1 word of static link for Pascal procedures: pointer to enclosing procedure's frame. Not used by C or FORTRAN.

f) 4 words of linkage area (reserved).

g) 4 words allocated for the first four words of **foo**'s incoming argument list.

foo can use the Store Multiple (stm) and Load Multiple (lm) instructions to save and restore registers, from any starting register through r15. r0-r5 never need to be preserved. Prolog and epilog examples below show how the caller's r1 is restored.

The floating register save area holds up to 4 doubleword registers ending with register 5. No space is allocated if no floating registers need to be preserved.

The file */usr/include/frame.h* gives symbolic definitions for the sizes and offsets of some of these areas.

3. Register Usage

Certain registers, such as r1, have specific uses throughout execution; others, like r15, are specified during a call and are free at other times. The following table defines register usage at the call interface.

Register	Preserved over call	Usage
r0	no	called function's data area pointer
r1	yes	stack pointer (to caller's frame)
r2	no	argument word 1 and returned value
r3	no	argument word 2 and second word of a returned double value
r4	no	argument word 3
r5	no	argument word 4
r6-r13	yes	register variables, etc.
r14	yes	data area pointer (not required)
r15	no	return address
mq	no	multiply/divide register

r1 always addresses the bottom of the stack frame of the currently executing function. A compiler may assign another register to address the high end of the stack frame. The portable C compiler, for instance, points r13 at the last 64 bytes of auto storage. The linkage convention requires this second register only for *alloca* support (see the section entitled **Alloca Storage Allocation** below). The register number and the offset from the frame top, which are arbitrary, are recorded in the trace table.

Floating-point registers 0, 1, and 6 are not preserved over a call. Registers 2 - 5 must be preserved. Floating point registers are not used to pass arguments or return results.

4. The Data Area

The data area (also called "constant pool," which is a misnomer) is addressed by r0 on entry to **foo**. The word pointed to by r0 must contain **_.foo**, the address of **foo**'s entry point. The following word supports the profiling option, and if present must be initialized to zero; the third word, also optional, supports *alloca* storage allocation.

It is conventional, but not required, for r14 to address the data area during execution. (The optional profiling linkage, which follows the prolog, does require it momentarily.)

For easy addressability, other data such as static variables, strings, or a literal pool may be located in the data area, either before or after the word addressed by r14.

A value **&foo** of type pointer to function corresponds to the address of **_foo**, the function's data area, not the address of **_.foo**, the function's entry point. A program that does arithmetic on function pointers, assuming that they address entry points, will probably malfunction.

5. Argument Lists

Arguments are word-aligned and allocated to consecutive stack locations. The list spans frame boundaries: words 1-4 are allocated in the top of the called function's frame, and the remainder are stored in the bottom of the caller's frame, which is adjacent. Argument words 1-4 are passed in registers r2-r5, not on the stack. The called function may choose to store them in the allocated stack locations, but this is not necessary except in a function like *printf* which accesses its argument list via a pointer variable. Such functions must use the *varargs*(3) macros to assure that argument words 1-4 get stored properly.

Arguments are passed as follows, based on argument type:

- An int is passed in a single word.
- A long, short, pointer, or char is treated as an int and passed in a word. A function pointer is represented by the address of the function's data area.
- A double is passed in two successive words.

- A float is converted to a double and passed in two successive words.
- A structure is word-aligned to a full word and left justified, except for structures of 1, 2, or 3 bytes, which are right justified.

If the function is declared to return a structure, the caller passes the address of a result area in r2, and word 1 of the explicit argument list is passed in r3. Subsequent argument words are shifted accordingly.

6. Calling Sequence

A typical call of a function **foo** first prepares the argument list, then executes the following:

```
balix   r15,_.foo          # call
l       r0,$.long(_foo)    # get its data area pointer
```

Dereferencing a function pointer calls a function without needing to know its name. Suppose that the function pointer, which addresses the function's data area, is in r8. A typical instruction sequence is:

```
ls      r7,0(r8)    # get address of entry point
balrx   r15,r7      # call whomever
mr      r0,r8       # r0 = data area pointer
```

7. Prolog

The prolog saves the caller's registers and obtains stack space for the stack frame. A typical instruction sequence is:

```
_.foo:   stm    r10,-60(r1)      # save caller's registers
         ai     r1,-framesize    # allocate our stack frame
         mr     r14,r0           # initialize data pointer
```

Other instruction sequences are needed for frame sizes larger than 32768 bytes. A sequence that decreases r1 in two stages is acceptable if the stack remains protected at all times. An example of an *un*acceptable sequence for a frame size of 64536 is

```
cal    r1,-bothalf(r1)    # -bothalf = 1000
cau    r1,-tophalf(r1)    # -tophalf = -1
```

This momentarily increases r1, letting an ill-timed interrupt destroy the stack.

8. Profiling

If either the -p or -pg option is selected, this instruction sequence follows the prolog and accomplishes data collection for performance monitoring:

```
mr     r0,r15
bali   r15,mcount     # r0  = caller's return address
                      # r14 = our data address
                      # r15 = our return address
```

9. Epilog

The epilog prepares a result, restores the caller's environment, and returns control. A typical instruction sequence is:

```
lis    r2,0                    # zero result returned in r2
lm     r10,framesize-60(r1)    # restore registers
```

```
        brx    r15                    # return
        ai     r1,framesize           # adjust stack frame
```

The location of the return value depends on the function type:

- An int, long, short, pointer, or char is returned in r2.
- A double is returned in r2 and r3.
- A float is returned as a double.
- A structure result is returned by moving it into the area pointed to by the first argument list word (in r2 on entry).

10. Alloca Storage Allocation

The implementation-dependent storage allocator *alloca* (see *malloc*(3)) expands its caller's stack frame by decreasing r1, to obtain a storage area that is automatically deallocated on return. The storage area so obtained starts at the end of the maximum-length argument list in the newly expanded frame. *alloca* can be called from any function that follows two conventions:

(1) It addresses outgoing argument lists through r1, and addresses all other areas in the stack frame through some other register (identified in the trace table as *frame_reg*).

(2) In its data area, which must be addressed by r14, the halfword at (r14)+8 holds the value 0xf690 (a magic number, used for validity checking). The halfword at (r14)+10 holds the length of the longest outgoing argument list (exclusive of the first four words, which do not occupy space in the frame).

Files compiled with the *hc*(1) or *pcc*(1) option **-ma** adhere to these conventions.

11. Trace Tables

Debuggers rely on a trace table of 6-10 bytes following the text of each function. A debugger locates a trace table by searching forward through program text (generally from a point indicated by a call's return address). The search stops when it finds two successive halfwords, each having 0xdf in its first byte. For compiled C functions, or assembler functions following the same conventions, the trace table corresponds to the following structure:

```
struct TT_D_COM {
unsigned magic1 : 8           = 0xdf,
         code  : 8            = 7,
         magic2: 8            = 0xdf,
         first_gpr : 4,
         optw  : 1            = 1,
         optx  : 1,
               : 1            = 0,
               : 1            = 0;
char     npars : 4,
         frame_reg : 4;
char     first_fpr : 4,       /* This byte present */
               :4;            /* only if optx= = 1 */
char     lcl_off_size : 2,    /* lcl_offset is variable length */
         lcl_offset1 : 6,
         lcl_offsetn[lcl_off_size];
         }
```

first_gpr is the first register saved by the store multiple instruction in the prolog. This indicates the size of the general register save area.

optx is 1 if the byte holding *first_fpr* is present; otherwise, it is 0.

npars is the number of words of declared arguments. The maximum value of 15 does not restrict the actual length of argument lists.

frame_reg identifies the register used to address local variables, etc., in the stack frame. *frame_reg* is 1 unless an alternative register is used, e.g. r13 for pcc−compiled functions.

first_fpr and the following 4 reserved bits are present only if *optx* is 1. Values 2, 3, 4, and 5 indicate that floating point registers 2-5, 3-5, 4-5, or 5 were saved in the floating register save area, and indicate the size of the save area. If *first_fpr* is 0 or not present, no floating registers are saved.

lcl_offset is an unsigned integer 6, 14, 22, or 30 bits long. It indicates the distance, in words, to the top of the stack frame from the point addressed by register *frame_reg*.

12. as(1) Routines

A very simple C function or hand-coded assembler function, that doesn't call other functions, can take some shortcuts. It may not need to save and restore registers, and need not allocate a stack frame if the protected area between (r1)−0x64 and (r1) gives it sufficient storage.

Such a function has a simpler trace table: the four byte sequence 0xdf02df00. Debuggers may not be able to backtrace from this function if the caller's r14 and r15 are disturbed.

Temporarily each file must include the lines:

```
        .globl   .oVncs
        .set     .oVncs,0
```

This is used by *ld*(1) to detect use of an obsolete linkage convention. Compilers generate definitions of .oVncs automatically.

13. Floating Point Arithmetic Linkage

The floating point linkage gives access either to the Floating Point Accelerator board, if it is present, or to the software emulator, if it is not. There is only one source and .o form; the same compiled program can be linked either "direct" to assume FPA presence, or "compatible" for hardware/software compatibility at a small decrease in performance.

The FPA is driven by loads and stores to addresses 0xff000000 and above. To use the floating point linkage, write each floating point load and store to use r2 for data and r3 to hold the FPA address. Expect that r0 and r15, and stack locations below (r1), may be destroyed. Then, in place of

```
        l r2,0(r3)
write
        .long FPaGET0
```

and in place of

```
        st r2,0(r3)
write
        .long FPaPUT0
```

Programs are linked by default for compatible access: FPaGET0 and FPaPUT0 resolve into calls on interface routines that use the FPA if it is present or the software emulator if it is not. To link a program for direct access, give *cc* or *ld* the option **-lfpa**. This resolves FPaGET0 and FPaPUT0 into load and store instructions, resulting in minimum overhead and a dependence on the FPA.

14. Addressability in Very Large Modules

When .o files are linked by *ld*(1), the resulting object module may be so large that the text of the caller and callee are more than 2^{20} bytes apart. The balix instruction in the call cannot then address the callee, and *ld* modifies the instruction in one of two ways to establish addressability.

A balax replaces the balix if it can duplicate the balix's effects, that is, the callee's address is below 2^{24} and r15 is the link register. Otherwise, the balix is replaced by a balix to a piece of "trampoline code" that derives the callee's entry point address from the contents of r0 and branches to it.

Other than in function calls, addresses are always carried as 32-bit values, so addressability is unaffected by module size.

REFERENCES

(1) Johnson, S. C. and D. M. Ritchie. "The C Language Calling Sequence," Computing Science Technical Report No. 102, Bell Laboratories, Murray Hill, NJ, 1981.

(2) "Assembler Reference Manual for 4.2/RT," in Volume II, Supplementary Documents.

(3) *IBM RT PC Hardware Technical Reference*, SV21-8024

This page intentionally left blank.

Recompiling with High C

ABSTRACT

Both *pcc* (the standard C compiler provided with Berkeley systems) and MetaWare High C are available in 4.2 for the RT PC. Although High C offers significant advantages over its predecessor, *pcc* remains the default C compiler. This article serves as a guide for C programmers in recompiling existing programs with High C. The article contains three chapters:

1. **Introduction** describes High C, contrasting it with *pcc*.

2. **Diagnostic Messages** explains a sample High C diagnostic message and describes messages frequently encountered when recompiling programs with High C.

3. **Run-Time Differences** describes those differences between *pcc* and High C that may not manifest themselves until run-time.

1. INTRODUCTION

4.2/RT now provides a new optimizing C compiler, MetaWare High C, in addition to the standard *pcc*-based C compiler. High C provides extensive code optimization, producing compiled programs that run up to twice as fast as *pcc*-compiled programs. It also generates tighter code; object file text is typically 15% smaller than with *pcc*.

Hc has been tested against the C Test Suite provided by Human Computing Resources Corporation, and is used to compile the entire 4.2/RT system (with the exception of assembler routines and a few other files).

The commands *hc*(1) and *pcc*(1) are available in the /bin directory. Users are not obliged to use one compiler or the other. The command *cc*(1) in /bin is a symbolic link that may point to either *hc* or *pcc*. In the 4.2/RT system as distributed, /bin/cc points to *pcc*.

The *hc* feature you will notice first is probably its meticulous semantic and syntactic checking and precise diagnostics. Many old programs that compile "error free" with *pcc* generate warnings and errors with *hc*, usually for good reason. In recompiling 4.2/RT, we found that messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized or misspelled variables that had been undetected for years. The "High C Programmer's Guide" tells how to use flags and toggles to adjust the error and warning sensitivity up or down; we recommend "up" during program development.

High C represents a significant step toward the draft ANSI C standard, and supports a more extended C language than does *pcc*. The *High C Language Reference Manual* describes the extensions in full. One extension that may affect existing programs is the presence of new keywords: **signed**, **const**, and **volatile** for ANSI, plus **pragma** (borrowed from Ada). The keyword **signed** is supported; **const** and **volatile** are reserved but not implemented due to the preliminary nature of the ANSI definition. A program using any of these four names for identifiers will have to be modified.

Two other ANSI-related changes, character escapes and widening rules, are discussed in the sections on "Character Escapes" and "Integer Widening" below.

In general, High C supports the semantics of "classical" C, where this is not precluded by adherence to the draft ANSI C standard. Even so, there are circumstances in which a language construct that is incompletely defined may execute differently when compiled with *hc* and *pcc*. Chapter 3, "Run-Time Differences," discusses constructs whose semantics may differ.

2. DIAGNOSTIC MESSAGES

This section provides an explanation of a sample diagnostic message and includes a list of diagnostics frequently produced when recompiling with *hc*. The list provides an explanation of each diagnostic and, where appropriate, a recommended solution.

2.1. Sample Diagnostic Message

The following shows a code fragment, a diagnostic message generated by the code, and an explanation of the message.

Code Fragment:

```
1       /* this file is named test.c */
2
3       main()
4
5       {
6        char *j;
7        int  i;
8
9        i = j + i;
10
11      }
```

Diagnostic Message:

```
E "test.c", L9/C5:
|    Type *Unsigned-Char (at "test.c", L6/C6) is not assignment compatible with type Signed-Int.
```

Explanation:

- The "E" stands for Error. Warning messages begin with a "w."
- "test.c" is the name of the module containing the error.
- L9/C5 indicates the error was detected in Line 9, Column 5.
- The body of the error message explains that a value (j + i) of type pointer to **unsigned char** was being assigned to a variable (i) of type **signed int**. This is illegal (but unchecked by *pcc*).
- The phrase (at "test.c", L6/C6) locates the declaration that gave rise to the value of type pointer to **unsigned char**. This is particularly helpful in locating declarations in #include files.
- The vertical bar "|" in the first column indicates a continuation line of a multiline message.

2.2. Common Diagnostic Messages

This section lists the most frequently encountered messages and suggests ways to resolve them. See the section on diagnostic messages in the "High C Programmer's Guide" for a complete list of warning and error messages.

Type t is not assignment compatible with type t′.

The mismatched-type message appears for any of several reasons. Most frequently, it has to do with pointer conversion, and can be eliminated by using explicit casts. In this example, the comments propose ways to rewrite each statement.

```
main()
{
        char *pc;
        int *pi, i, x;

        pc = pi;            /* should be:  pc = (char *)pi; */
        x = pc + i;         /* should be: x = (int)(pc + i); */
        i = pc;             /* should be: i = (int)pc; */
}
```

Another common cause of this message is shortcuts in structure initialization. As an example, given the declaration:

```
struct s1 { int i, j; };
```

the shortcut initialization:

```
struct s1 x = 0;
```

is allowed by *pcc*, but C syntax (and *hc*) require braces around the initializer:

```
struct s1 x = {0};
```

Variable is set but is never referenced.

This message warns of an initialized variable that is not used in the module. It may be a symptom of a logic error.

This diagnostic prints in another common situation: if RCS or SCCS variables are contained in the program header. In this case, you can ignore the message.

Result of comparison never varies.

An expression was found whose operands are such that the value of the expression is always the same. The usual cause is a logic error arising from confusion over signed/unsigned types. For example, an **unsigned char** is never negative; therefore, a comparison with a negative constant will never vary. Look for assumptions that the type **char** is really signed.

Variable required.

This generally points out an illegal left-hand side of an assignment. This error can be produced by statements of the form:

```
(CONDITION ? i : j) = -1;
```

which *pcc* (incorrectly) allows if CONDITION involves only constants and preprocessor variables. Rewrite it as:

```
*(CONDITION ? &i : &j) = -1;
```

This is multiply-declared.

This may be the result of a variable declared **extern**, then redeclared later in the same module as **static**. This is often caused by an **extern** declaration in an #include file. *Pcc* allowed the redeclaration. Correct this by using distinct names for the two variables.

Local function is never referenced; no code will be generated for it.

A function of storage class **static** is not called anywhere in the compilation unit. Since it is not exported, there can be no reference to the function, and it is eliminated as dead code. The -g option disables this optimization, so that *dbx*(1) sessions can access such functions.

Expression has no side effect and has been deleted.

The value of an expression is not assigned to a variable or otherwise used to affect the computation. For example, "2+3;" is useless and is deleted.

This function declaration is inconsistent with the "int"-returning function declaration imputed at Ln/Cm.

A function that is called before it is declared is assumed to return **int**. Any subsequent declaration of the function must declare it to do so.

Correct this by placing an explicit declaration of the function with the proper return type before the first call (and check all calls for their assumptions about the return type!).

Unexpected char.

Pcc allows multi-character character constants; *hc* does not. For example, for the following declaration:

```
int x = 'abcd';
```

pcc assigns the value 0x61626364 to *x*, but *hc* generates the above error message.

Fewer arguments given than function has parameters.

Hc checks argument lists in calls of functions that are declared in the same module.

3. RUN-TIME DIFFERENCES

Some of the differences between *hc* and *pcc* will not manifest themselves until load- or run-time. This chapter describes these differences and provides an explanation for their causes.

3.1. Order of Execution

C semantics permit subexpressions in a larger expression to be evaluated in any order, or even concurrently. The statements

```
i = j + j++ ;
foo( i, i--);
```

do not have well-defined meanings and may well execute differently with *hc* and *pcc*. To assure that side effects like assignment occur in a defined sequence, break such expressions into multiple statements.

3.2. Multiple Assignments

Look out for multiple assignments that require both narrowing and widening integer values, such as:

```
int i; char c;
i = c = integer-expression;
```

Here the integer-expression is "narrowed" on assignment to c. Language rules require (and *hc* supports) assignment of the narrowed value to i, not the original value. Code generated by *pcc* often fails to narrow the value correctly, and some incorrect programs may execute as intended only because of this *pcc* bug. Reorder the assignments, or write two statements.

3.3. Keyword "asm" Not Supported

Pcc allows inclusion of assembler statements within C programs via the "asm" construct. As *hc* does not produce intermediate code and generates code which is optimized across statements, this keyword is not supported.

Existing code which contains "asm"s will generate errors at load-time, with "_.asm" and "_asm" as unresolved references.

3.4. Volatile Memory

Hc optimizes the following code:

```
if (*p == 0) buf = *p;
```

by loading the contents of location p into a register for the comparison, then using this same register for the assignment as well. If p is the address of memory that is volatile (for instance, it is an I/O register that is updated after each reference), the assignment will not reflect the changed value. Since this type of code is common in device drivers (and other portions of the kernel), *hc* provides a flag (**-Hvolatile**) which disables common subexpression recognition across statement boundaries.

3.5. Use of *setjmp*(3) and *longjmp*(3)

Code that uses *setjmp* and *longjmp* sometimes makes assumptions about the values of **auto** variables. *Hc* may classify frequently referenced **auto** variables as **register** variables, which generally only improves execution speed and code size. However, **register** variables may not have their most current value after *longjmp* returns to *setjmp*'s point of call. (This is linguistically acceptable, but can still be a nuisance.)

You can inhibit allocation of an **auto** variable v to a register by using the expression &v anywhere in the function. A broader approach is to use the auto-reg-alloc compiler toggle. Specifying **-Hoff** = **auto-reg-alloc** on the command line prevents *hc* from classifying **auto** variables as **register** variables. This toggle can also be turned off around a particular function via the pragma statement, so that the rest of the module remains fully optimized.

3.6. Character Escapes

Hc supports the draft ANSI complement of character escapes:

\a alert (bell)	\t horizontal tab
\b backspace	\v vertical tab
\f form feed	\x*nnn* hexadecimal numeric
\n newline	\' single quote
\r return	\" double quote

Use of an undefined character escape results in a warning message.

3.7. Integer Widening: Value-Preserving vs. Unsignedness-Preserving

Historically, C compilers have used either of two widening rules: unsignedness-preserving (u-p) widens an unsigned **char** or **short** to **unsigned int**; value-preserving (v-p) widens it to a **signed int**. U-p is sometimes useful but creates many anomalous situations. Note the following example.

```
void f ()
{
     unsigned char c = getchar ();

     if (c - '0' < 0 || c - '0' > 9)
        printf("This character is not a digit");
}
```

Because *pcc* uses the u-p rule, the test (c - '0' < 0) will always fail (since an **unsigned int** can never have a value less than 0). Because *hc* uses the v-p rule, c will be widened to a signed integer; the test will work as expected. The v-p rule almost always produces the expected result, and is the rule chosen by the ANSI committee in the draft standard.

3.8. Size of Enumerated Types

Hc has a much cleaner implementation of enumerated types, within ANSI rules. Sizes differ between *pcc* and *hc*. In *pcc*, the size of enums is 4 bytes; in *hc*, 1, 2, or 4 bytes. Incompatibility can occur only if two modules sharing the same enum quantity are compiled with different compilers.

References

- Appendix C of this manual, which contains the "High C Programmer's Guide"
- *High C Language Reference Manual*, available from:

  MetaWare Incorporated
  903 Pacific Avenue, Suite 201
  Santa Cruz, CA 95060
  (408) 429-META
- Draft proposed American National Standard for the C Language; contact ANSI Committee X3J11 for the most recent draft.

This page intentionally left blank.

Professional Pascal Differences

ABSTRACT

Professional Pascal is available as an option with Release 2 of 4.2/RT. Professional Pascal offers significant advantages over other Pascal compilers. It is a highly optimizing Pascal compiler that conforms to the ANSI Standard, and includes many useful extensions, such as support of varying length strings, bitwise operations, packages, and iterators.

This article points out the major differences between Berkeley Pascal and Professional Pascal, as an aid to programmers recompiling existing programs with Professional Pascal. The article has two chapters:

1. **Introduction** describes Professional Pascal, contrasting it with Berkeley Pascal.

2. **Significant Differences** briefly describes those differences that may prevent a program that compiles with Berkeley Pascal from compiling (or executing correctly) with Professional Pascal.

1. Introduction

Pascal programs which are not dependent upon a particular compiler's extensions, that is, programs written in ANSI Standard Pascal, should port to 4.2/RT using Professional Pascal[1] with little or not effort. However, programs written in Berkeley Pascal may not port so easily. Berkeley Pascal includes many extensions to standard Pascal (which are outlined in Appendix A, "Appendix to Wirth's Pascal Report," of the "Berkeley Pascal User's Manual" in *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, Volume 2C*). If a program uses any of these extensions, it may not compile or execute as expected.

This article concentrates on the features of the Berkeley Pascal compiler[2] that are missing or differ from Professional Pascal. The article does not attempt to point out the many features of Professional Pascal which are not found in Berkeley Pascal. For a complete description of Professional Pascal extensions, please see *Professional Pascal Language Extensions Manual with Rationale and Tutorials*, available from MetaWare Incorporated.

2. Significant Differences

Several differences exist between *pp* and *pc* which may affect your programs. This section points out these differences.

2.1. Case of Identifiers

In *pc*, the names of identifiers are case-sensitive. In *pp*, they are case-insensitive; *all names* are shifted to lower case. Be sure all identifiers are uniquely named regardless of case.

2.2. In-Line Compiler Directives

Pc supports in-line control of compile-time options from within comments:

```
{$option}
```

Pp provides similar support of "toggle" setting via pragma statements. See "Compiler Toggles" in the *Professional Pascal Programmer's Guide.*

2.3. Octal Constants:

In *pc*, an integer constant is expressed in octal by a series of digits terminated with "B" or "b" (e.g. 777b). *Pp* precedes the digit series with the character string "8#" (e.g. 8#777).

2.4. New Reserved Words

In addition to standard Pascal keywords, the words **pragma**, **package**, **iterator**, **value**, and **otherwise** are reserved in *pp*. (They are not reserved in *pc*.)

2.5. Predefined Routines

The following predefined routines found in *pc* are not supported in *pp*:

- Predefined procedures: *date*, *flush*, *linelimit*, *message*, *null*, *pack*, *remove*, *stlimit*, *time*, and *unpack*.
- Predefined functions: *card*, *expo*, *random*, *seed*, *sysclock*, *undefined*, and *wallclock*.

The routines *argc* and *argv* are not predefined as they are in *pc*, but they are defined in the "arg" package provided with *pp*. Note, however, the slightly different semantics for these

[1]Hereinafter referred to as *pp*.

[2]Hereinafter referred to as *pc*. Note that what is true for *pc* in this article is also true for *pi*, the Berkeley Pascal interpreter. Therefore, *pc* can be taken to mean "*pc* and *pi*."

routines as they are defined in "Utility Packages" in the *Professional Pascal Language Extensions* manual.

Similarly, the *clock* function is not predefined but is included in the *pp* "system" package.

The procedure *halt* is predefined in *pp* (as it is in *pc*), but it does not produce a control flow backtrace upon termination.

2.6. Writing Expression in Octal or Hexadecimal

In *pc*, the value i is displayed in octal by:

```
write(i oct)
```

or in hexadecimal by:

```
write(i hex)
```

where i is a **boolean**, **char**, **integer**, pointer or enumerated type. In *pp*, the equivalent would be:

```
write(ord(i):n:8)
```

or:

```
write(ord(i):n:16)
```

where "n" is the minimum field width.

2.7. Reading and Writing Enumerated Types

Reading and writing of enumerated types is not allowed in *pp*.

2.8. Associating File Name and Variable Name

In *pc*, a global file variable appearing in the **program** header is associated with a physical file of the same name. In *pp*, file variables appearing in the **program** header are associated with file names appearing as command-line arguments. See "Invoking the Compiler" in the *Professional Pascal Programmer's Guide.*

2.9. No Assert Statement

The **assert** *statement of pc* is not supported in *pp*.

2.10. Relational Operators on Sets

The relational operators " < " and " > " may not be applied to sets in *pp* as can be done in *pc*.

2.11. Simple Types Integer and Real

In *pc*, an **integer** is 32 bits wide. That is, it follows the conceptual definition:

```
type integer =  −2147483648..2147483647;
```

In *pp*, an **integer** is 16 bits wide; it follows the conceptual definition:

```
type integer =  −32768..32767;
```

Pp predefines the type **longint** to represent 32-bit integers; it is equivalent to *pc*'s type **integer**.

Pc represents a **real** in double-precision, or 64 bits. *Pp* represents **real** in single-precision, or 32 bits. *Pp* predefines the type **longreal** to represent double-precision; it is equivalent to *pc*'s type **real**.

If a *pc* program which is dependent on 32-bit **integers** and double-precision **reals** is ported to *pp*, the following redefinitions can be used:

```
type
     integer = longint;
     real = longreal;
const
      maxint = maxlong;
```

2.12. Predefined Types

Pc predefines the types **alfa** and **intset** as:

```
type
     alfa = packed array [1..10] of char;
     intset = set of 0..127;
```

These types are *not* predefined in *pp*; the above definitions can be added to existing programs that depend upon these types.

2.13. Subrange Mapping

In *pc*, the subrange 0..255 is mapped to a 16-bit word. In *pp*, it is mapped to an unsigned byte.

Pc maps the subrange 0..65535 to a 32-bit longword; *pp* maps it to an unsigned (16-bit) word.

2.14. Global Variables

In *pc*, all variables at the outermost level are made global static. In *pp*, such variables are made local static by default. The preferred way to share variables across modules in *pp* is via interface packages; however, the statement "pragma data(COMMON);" can be specified before the first variable declaration to achieve the same effect from *pp*.

2.15. Predefined Constants

Pc predefines the integer constant "minint"; *pp* does not. The following definition can be used:

```
const
     minint = −maxint−1;
```

The predefined character constants "minchar," "maxchar," "bell," and "tab" of *pc* are not supported in *pp*.

References

- Appendix C of this manual, which contains the "High C Programmer's Guide"
- *Professional Pascal Documentation Set*, available from:

  MetaWare Incorporated
  903 Pacific Avenue, Suite 201
  Santa Cruz, CA 95060
  (408) 429-META

4.2/RT Console Emulators

This paper explains the need for, and design of, console emulators in 4.2/RT. It contains the following sections:

1. OVERVIEW

This emulator package was developed under 4.2/RT to support the complex data streams characteristic of advanced workstations. Traditional line disciplines and console driver interfaces are not powerful enough to manage elaborate bit-mapped displays. Sophisticated keyboards, mouse devices, and multiple consoles add complexity to console management.

Emulators written using the 4.2/RT emulator package can handle normal line-discipline I/O functions as though they were normal *tty* hardware drivers. The emulator package also supports window-manager device-dependent routines, allowing an emulator to act as an interface to a window-manager/graphics system and to control screen output.

1.1. Bit-Map Terminal Requirements

Bit-map terminals are bit-addressable; they deal not with characters, but with individual bits. To represent a single character on the screen, a bit-map terminal turns bits at different locations on or off. In scrolling, all bits on a bit-map screen move up a line at a time, while bits on the bottom line turn off. Some bit-map terminals handle some or all of this in hardware. However, to act as a normal *glass tty* console, each terminal must also be able to perform standard I/O operations. Supporting multiple display types requires code to emulate a glass tty on each display without reproducing the same code for each.

An emulator package solves the inability of bit-map displays to deal directly with characters. It provides a standard interface needed by the low-level, device-dependent drivers and called by the higher-level line disciplines. The device-dependent drivers contain functional procedures that determine where a character appears on the screen, while the device-independent line disciplines determine what the character looks like. When requested to display information on the screen, an emulator package works between the two to ensure that the correct character appears at the correct location.

1.2. Output Emulators

Emulator code intercepts characters and analyzes them according to the type of tty emulated, then calls the appropriate routines to operate on the display. Using this design, any emulator works on any display on the workstation without knowing anything about the display. This type of emulator is an *output emulator.*

1.3. Input Emulators

The same design approach applies to input emulation. Any keyboard or mouse on the system must be able to pass data to the user in a given format. An *input emulator* accepts and deciphers data appropriately.

2. EMULATOR PACKAGE FUNCTIONS

The emulator package performs the following functions:

- Initializes each display present on the workstation.
- Provides a default emulator, defined for any particular hardware. An application does not have to choose an emulator at start-up.
- Allows multiple displays to run on the system simultaneously. Each display can be associated with a different process. This allows a separate login to run on each display.

The emulator package also allows the user to:

- Decide which display should be the default on system boot.

- Reinitialize any hardware or emulator.
- Select from a set of existing emulators for a display.
- Switch between the displays currently open on the workstation. When you switch screens, the focus of the keyboard and mouse moves to that display. This allows you to run a different window system on each display and press a hot-key to switch between them.
- Find and change the hot-key.
- Lock out the user or the system from a display. User lockout is useful for window managers that want to keep other applications from taking over the screen. Kernel or system lockout is useful when you don't want the kernel to attempt to use a non-existent display; for example, when an adapter has no display attached to it.

3. OUTPUT EMULATOR INTERFACE

An emulator needs general information about the display it uses, such as the number and width of lines that fit on the screen in the current font.

The following device-dependent procedures support any basic output emulator:

(1) Determine whether the display is present.

(2) Initialize the display.

(3) Position the cursor anywhere on the display.

(4) Display a character at the cursor position by putting up a bitmap from an internal data font.

(5) Blank a given section of the display (by character).

(6) Move a group of lines on the display.

(7) Print screen contents on the standard printer.

The following structure from *<machinecons/screen_conf.h>* describes the interface between the emulator and the display-dependent routines. From the top down to *flags* is the standard glass tty information; other entries are described later. This structure is initialized in */sys/machinecons/screen_conf.c*.

```
struct screen_sw {
        char    *name;                  /* Name of display */
        int     (*probe)();             /* Probe for screen */
        int     (*init)();              /* Initialize screen */
        int     (*s_putc)();            /* Put character on screen */
        int     (*pos_cur)();           /* Position cursor on screen */
        int     (*blank)();             /* Blank a section of screen */
        int     (*move)();              /* Move some lines on screen */
        int     (*printscreen)();       /* Routine to print screen */
        char    *rwaddr;                /* Read & writable addr on screen */
        short   lines;                  /* Number of lines on screen */
        short   width;                  /* Width of screen in characters */
        short   vbits;                  /* Vertical number of screen bits */
        short   hbits;                  /* Horizontal number of screen bits */
        int     flags;                  /* Some flags about the screen */
        int     def_oute;               /* Default output emulator */
        int     (*pos_loc)();           /* Position locator on screen */
        int     (*load_loc)();          /* Load locator description */
        int     (*show_loc)();          /* Show locator on Screen*/
        int     (*hide_loc)();          /* Hide locator on Screen */
        int     (*apa_init)();          /* All points addressable screen init */
        int     fill_int[14];           /* Fill out to 32 ints (2^5) */
};
```

The synopsis below from *screen_conf.h* shows the interface to the above structure. The emulator need only use the following routines and attributes:

```
/* General Defines for emulators to use with the screen_sw structure */
#define WS                    si->which_screen
#define SCREEN_LENGTH    (screen_sw[WS].lines)
#define SCREEN_WIDTH     (screen_sw[WS].width)
#define SCREEN_SIZE(SCREEN_LENGTH * SCREEN_WIDTH)
#define STATUS_LINE          (SCREEN_LENGTH - 1)

/* Character Attributes */
#define NORMAL_VIDEO     0x01
#define REVERSE_VIDEO    0x02
#define UNDERLINE_VIDEO 0x04
#define HI_INTENSITY          0x08
#define BLINK          0x10

/* Defines for calling console screen dependent switched routines */

/* Put character 'c' with attribute 'screen_attr' on console */
#define screen_putc(c, screen_attr) (*screen_sw[WS].s_putc) (c, screen_attr)

/* Move cursor to x,y position */
#define pos_cursor(x, y) (*screen_sw[WS].pos_cur) (x, y)

/* blank with screen_attribute from start coordinates to end coordinates */
#define screen_blank(s_a, sy, sx, ey, ex) (*screen_sw[WS].blank) (s_a, sy, sx, ey, ex)

/* Macro for blanking a line */
#define blank_line(s_a, line) screen_blank(s_a, line, 0, line, SCREEN_WIDTH-1)

/* move line1 ... line2 to dest */
#define screen_move(l1, l2, dest) (*screen_sw[WS].move) (l1, l2, dest)

/* Position screen locator on screen at x,y position with msbox restriction */
#define pos_locator(x, y, msbox) (*screen_sw[WS].pos_loc) (x, y, msbox)

/* Load a new screen locator description with msbox restriction */
#define load_locator(c, msbox) (*screen_sw[WS].load_loc) (c, msbox)

/* Show screen locator with msbox restriction */
#define show_locator(msbox) (*screen_sw[WS].show_loc) (msbox)

/* Hide screen locator */
#define hide_locator() (*screen_sw[WS].hide_loc) ()

/* APA Screen init */
#define apa_initialize() (*screen_sw[WS].apa_init) ()
```

4. INPUT EMULATOR INTERFACE

The low-level interface to the input emulator is not defined as strictly as the one for the output emulator. Basically, a set of hardware routines in *keyboard.c*, *kls.c*, *speaker.c*, and *mouse.c* can be called by an input emulator to control the keyboard, speaker, and mouse. The input emulator receives a data interrupt from the keyboard or mouse. The emulator deciphers the data and tracks the state of the device; then passes its processed data to the user through a *line discipline* or some other emulator-specific method, such as shared memory.

Few procedures are needed to control a keyboard for setting the auto keyclick rate, bell tone, and key characteristics (repeat rate, make/break, etc.). Because few workstations support multiple keyboards simultaneously, there is no need to set up a switch table for these hardware routines. Workstation mouse devices also have few control operations (set sampling or resolution rate); input emulators do not yet deal with these operations directly, but instead pass *ioctl* system calls to the appropriate driver.

5. WINDOW MANAGER DEVICE-DEPENDENT ROUTINES

Listed below are the device-dependent routines available with the *screen_sw* low-level routines. These are normally used in an input emulator to control the graphics cursor.

pos_loc()
: Position the locator at a given coordinate on the display.

load_loc()
: Load a locator bitmap for the display. This is the locator until the next load_loc.

show_loc()
: Make the locator visible and keep showing when positioned. Usually used after a hide_loc.

hide_loc()
: Make the locator invisible, but do not affect the tracking.

apa_init()
: Initialize the display for graphic operations needed by the locator. Useful for displays with hardware cursors/locators.

6. SYSTEM INTERFACE TO THE EMULATOR

A new emulator should be easy to add to a system and must be able to coexist with other emulators. An emulator has many functions and system entry points similar to those of a tty hardware device driver. The main difference is that emulators funnel through a single console driver and call a common set of hardware routines, while device drivers deal directly with the hardware.

The emulator switch table below lists all routines necessary for a device driver to interface with an emulator. To add an emulator to the system, add the following routines in the switch table structure shown below. This structure (modeled after line disciplines) is declared in *screen_conf.h*, and the table is initialized in *screen_conf.c*.

```
/*
 * Emulator line control switch.
 */
struct emulsw
{
        int     (*e_open)();
        int     (*e_close)();
        int     (*e_read)();
        int     (*e_write)();
        int     (*e_ioctl)();
        int     (*e_rint)();
        int     (*e_putc)();
        int     (*e_select)();
        int     (*e_putstatus)();  /* to put up status information */
        int       fill[7];
};
```

Each emulator, depending on its needs, has the following entry points in the kernel:

e_open()
: Open the emulator to do any necessary initialization. Perform initial operations such as clearing the screen, initializing the cursor, and positioning the cursor on the screen.

e_close()
: Close the emulator; do any cleanup necessary.

e_read()
: Read data from the emulator (used only by input emulators). For most emulators, this routine forwards the read request to the user-defined line discipline. The read routines in line disciplines currently perform the operations necessary for this routine. This consists of taking the already-received characters off a *clist queue* and passing them to the user program's read buffer.

e_write()
: Write data to the emulator (used only by output emulators). This procedure takes a character stream passed from the user-level program. Most emulators call the line discipline specified by the user to do any character preprocessing. Again, the line discipline routines already perform the necessary duties for this routine. This routine and the e_read() routines are in the emulator package for completeness and to allow flexibility. Some specialized emulators do use these routines for other than calling the associated line-discipline routines (see *buf_emul*(4)).

e_ioctl()
: I/O control to emulator for changing or setting characteristics of the emulator or performing operations that do not fit into the normal interface to the emulator. This routine should return a (− 1) if the command is not recognized.

e_rint()
: Receive interrupt to emulator (used only by input emulators). The emulator receives an interrupt from a driver's interrupt routine and processes the data depending on the type of interrupt received. This procedure passes the processed input data to the user-assigned line-discipline input routine. Some specialized window-manager emulators do not forward these data to a line discipline, but do their own queuing and interacting with a window manager.

e_putc()
: Put a character on the display (used only by output emulators). This emulator routine receives a character from a user's write or kernel printf. The emulator deciphers the data and interprets character strings before passing the appropriate characters to the hardware putc routine. This procedure makes use of the screen switch table (screen_sw) in calling the device-dependent routines to perform the emulation on any display.

e_select()
: Select call to emulator (used only by input emulators). This routine is used to perform

the normal select duty of informing the user process when new data are ready.

e_putstatus()
: Put status call to emulator (used only by output emulators). The emulator takes the passed string and places it at an offset on the status line.

7. CONSOLE DRIVER'S RELATIONSHIP TO 4.2/RT

The following diagram shows how the parts of the system described relate to the standard parts of a 4.2/RT system:

Console System Diagram

User Application		User Level
System Call Interface	Event Queue in Shared Memory	
Line Discipline Package	Emulator Package	Kernel Level
Standard Device Drivers	Low Level Display Dependent Routines	
Displays/Keyboard/Speaker System or Serial Mouse		Hardware

The above is a conceptual view of the system. It does not show all parts and interfaces, but indicates the levels of flow. The console driver routes normal driver requests to the correct display and input/output emulator, depending on the minor device specified.

The following shows how the minor device number maps to an emulator and display:

Output Emulator Flag	Bus	Display#
bits 7 - 4	bit 3	bits 2 - 0
0 or 1	0 or 1	0 - 7

The following is a list of currently-used displays supported by 4.2/RT:

Console Displays

Display #	Symbolic Name	Description
0	CONS_GEN	Generic console (current display)
1	CONS_AED	ACIS experimental display (stream ordered)
2	CONS_APA16	IBM 6155 Extended Monochrome Graphics Display (bitmap)
3	CONS_APA8C	IBM 6154 Advanced Color Graphics Display (bitmap)
4	CONS_APA8	IBM 6153 Advanced Monochrome Graphics Display (bitmap)
5	CONS_MONO	IBM 5151 Monochrome Display (character driven)

If the bus bit is set, opening the device grants access to the I/O bus. Without this bit, it is necessary to open */dev/bus* to gain access to I/O space. This bit is provided for compatibility; new applications should open */dev/bus* if they need bus access.

The emulator field in the minor device number tells the console driver that the default glass_tty input/output emulators will be used (0), or indicates that a non-standard output emulator will be used (nonzero). If a non-standard output emulator is used, the system restores the display to the standard state (*default emulators*) when the device is closed.

The following is a list of emulators currently available:

Emulators Available		
Emulator #	Symbolic Name	Description
0	E_KBDINPUT	Intelligent keyboard mapping input emulator (standard)
1	E_STDOUTPUT	Standard output emulator
2	E_IBMOUTPUT	IBM 3101 output emulator
3	E_ANSIOUTPUT	ANSI output emulator (not implemented)
4	E_XINPUT	X event queuing input emulator
5	E_BUFOUTPUT	Buffering output emulator
6	E_AED	Raw AED microcode interface emulator

The *default emulators* for each display are:

Default Input/Output Emulators for each Display		
Display	Input Emulator	Output Emulator
AED	E_KBDINPUT	E_STDOUTPUT
APA16	E_KBDINPUT	E_IBMOUTPUT
APA8C	E_KBDINPUT	E_IBMOUTPUT
APA8	E_KBDINPUT	E_IBMOUTPUT
MONO	E_KBDINPUT	E_IBMOUTPUT

7.1. Input From Keyboard Scenario

(1) User types character on keyboard.

(2) Receive interrupt in keyboard driver, keyboard.c, interrupt routine *kbdint()*. This routine extracts key code from the hardware.

(3) Call emulator receive-interrupt routine from the switch table indexed by the current *input focus* after setting the emulator structure flag, indicating that this was a keyboard interrupt.

(4) Emulator checks whether this was a keyboard interrupt. If so, it either translates code into a character and calls normal line-discipline routine for this console with the translated character, or performs some emulator-specific function such as storing the raw key code in a shared-memory area (X-like) and setting a semaphore, also in shared memory, to inform the user process that a new event has arrived.

(5) If a line-discipline input routine is called, it performs its previously-described normal input (editing/mapping) and passes the result to the user through the read system call interface.

(6) If a shared-memory queue interface is used, the user process notes the queue update through the semaphore and proceeds to read the data from the shared memory without performing a read or any other system call.

(7) In either of the above two cases, a select would be satisfied if the user had previously done a select call.

7.2. Output To Display Scenario

This scenario applies only to glass-tty operations. The window-manager system goes directly to the display hardware through its own graphics routines. If a user tries to write through the system to the display while a window manager is controlling the display, a special *buffer emulator* is called instead of a glass-tty emulator, as in the scenario below.

(1) The user performs a *write* system call with a buffer of data for the display. These data consist of ASCII data or display order streams.

(2) The console write routine then calls the write routine of the output emulator selected by the minor device number.

(3) For most emulators, the output-emulator write routine then forwards these data to the line-discipline write routine specified by the user. Certain emulators, such as the buffer emulator, intercept these data and capture them for printing later.

(4) The line discipline interprets the data and calls the console start routine to print the ASCII characters.

(5) The console driver's start routine loops through dequeuing each character and calling the output emulator put-character routine, *e_putc,* for each character.

(6) The output emulator put-character routine then deciphers the data and calls the appropriate device-dependent routine to display the character or perform the display command.

8. USER INTERFACE TO THE EMULATORS

The user interface to the emulators consists of system calls to the console driver (see *cons*(4) and *mouse*(4)).

8.1. Interface to Keyboard Input and Display Output

8.1.1. Standard Interface

In the simplest case, a user program still performs the same operations as in the past, allowing previously-written programs to work without change. The following lists the normal scenario and what the emulator package does:

Standard Console Interface Device				
Permissions	owner	major	minor	device
crw-rw-rw-	root	0	0	/dev/console

(1) An application such as login opens */dev/console*. Since */dev/console* is the special CONS_GEN minor device, in "open" the output is mapped to the current console-focused display. The display at which a program is started is the display associated with the process. The system starts the default input and output emulators for that display. Input is received only if the input focus is set to CONS_GEN.

(2) The application reads or writes to the file descriptor returned from the open system call. The system maps writes to the CONS_GEN minor device to the display with the current input focus. This causes the appropriate display-indexed input/output emulators to be used. Input is focused to CONS_GEN if no console tty devices and no console graphic devices are open, except when using the default input emulator (E_KBDINPUT). If the input focus is not on CONS_GEN, the input focus follows the output focus.

(3) The application exits or closes the */dev/console* file descriptor. The system closes the input/output emulators and the stream for the display with the current input focus.

Mapping */dev/console* to the current display is important for most applications that do not need to know on which display they are running. This mapping is also important at system bootup time where the single-user shell does not know on which display it is starting and is simply mapped to what the system chooses as the starting input focus. But some applications (for example, login, window manager) need to know on which display they should start. Therefore, the following devices are provided to support the displays available on 4.2/RT:

Standard TTY-like Display Devices				
Permissions	Owner	Major	Minor	Device
crw-rw-rw-	root	0,	1	/dev/ttyaed
crw-rw-rw-	root	0,	2	/dev/ttyap16
crw-rw-rw-	root	0,	3	/dev/ttyap8c
crw-rw-rw-	root	0,	4	/dev/ttyapa8
crw-rw-rw-	root	0,	5	/dev/ttymono

To start an application on a particular display, reassign its standard input and outputs to any of these devices or have the application specifically open one of them. The system routes output from the application to the appropriate display and its default emulators. Input to the application from the keyboard only occurs when the console focus is assigned to that display. To switch between open displays, press the specified hot-key for your system or use an application which performs an *ioctl* system call to set the input focus.

The different displays on the system are specified in the */etc/ttys* file that tells the system to start logins on each of the displays. A user can hot-key to the desired display and, after logging in, run any application needed. Any application started on that display stays associated with it, because it was started while the console focus was on that display. Because this is an application-transparent mapping to that display taking place in the kernel, a user can log on simultaneously to as many displays as needed.

8.1.2. Nonstandard Interface

For applications that call for a specific non-default emulator, the following devices are provided:

Nonstandard Display Devices				
Permissions	Owner	Major	Minor	Device
crw-rw-rw-	root	0,	65	/dev/aed
crw-rw-rw-	root	0,	66	/dev/apa16
crw-rw-rw-	root	0,	67	/dev/apa8c
crw-rw-rw-	root	0,	68	/dev/apa8
crw-rw-rw-	root	0,	69	/dev/mono

The emulator flag is nonzero for each of these devices. This indicates to the system that a nonstandard input and/or output emulator is going to be used on this display and that, on close, the system should return the display to its default emulators. The system opens the standard input emulator and a special buffering output emulator, because most applications that open the nonstandard device take over the screen and want output from other sources (such as kernel *printfs*) to be buffered and displayed when the display is closed. See *bufemul*(4) for more information. If bus access is required on open, add 8 to each minor device number.

The following is a list of commands available through the *ioctl* system call to the console emulator package:

Ioctl Commands to Emulator Package

Command	Read	Write	Description
CON_SELECT_SCREEN	Yes	Yes	Output focus is set to display number (arg > 0) or to next display in list (arg < 0). Previous display number is returned.
CON_GET_SCREEN	Yes	No	Just returns the current output focus display number.
EIGETD	Yes	No	Get the number of the current input emulator for this display.
EOGETD	Yes	No	Get the number of the current output emulator for this display.
EISETD	Yes	Yes	Set the input emulator and return the previous for this display.
EOSETD	Yes	Yes	Set the output emulator and return the previous for this display.
CON_INIT_SCREEN	No	Yes	Initialize the specified display (arg > = 0) or this display (arg < 0).
CON_GET_FOCUS_CODE	Yes	No	Get the current keyboard code for setting the console focus (xemul only).
CON_SET_FOCUS_CODE	Yes	Yes	Set the current keyboard code for setting the console focus (xemul only), and return the previous code.
SCRIOCGETF	Yes	Yes	Get screen control flags for the given display number.
SCRIOCSETC	Yes	Yes	Set screen control flags for the given display number.

All of the above commands take integer arguments except the last two. SCRIOCGETF and SCRIOCSETC use the following structure:

```
struct screen_control  {
                              int     device;  /* which screen/display to control */
                              int     switches; /* Flags for this screen */
};
```

Flags for Each Display

Flag	Description
CONSDEV_PRESENT	Display is present on this system.
CONSDEV_KERNEL	Display is available to the kernel.
CONSDEV_USER	Display is available to the user.
CONSDEV_INIT	Display has been initialized for output.
CONSDEV_TTY	Tty display has been opened directly by minor device number.
CONSDEV_GRA	Graphics display has been opened directly by minor device number.

All of the above *ioctl* system calls are device-independent controls for dealing with the emulators.

Each emulator has its own set of *ioctls* for its own emulation purposes. These other *ioctls* are used in window-manager emulators for operations such as passing/positioning the mouse locator for/on the display. See the man page for any particular emulator for more information.

8.2. Mouse Input Interface

The interface to the system mouse is similar to that of the keyboard. If the generic mouse device */dev/mouse,* minor device 0, is opened, the mouse input is attached to the display which has the current input focus. Opening any other mouse device attaches the mouse input stream to the process only when the input focus is on the associated display.

The following mouse devices are provided:

Mouse Input Devices

Permissions	Owner	Major	Minor	Device
crw-rw-rw-	root	15,	0	/dev/mouse
crw-rw-rw-	root	15,	1	/dev/msaed
crw-rw-rw-	root	15,	2	/dev/msmono
crw-rw-rw-	root	15,	3	/dev/msapa8
crw-rw-rw-	root	15,	4	/dev/msapa16
crw-rw-rw-	root	15,	5	/dev/msapa8c

The *system mouse* driver is essentially the same as those in other 4.2BSD-based systems. This driver hooks into the emulator package by selecting a special line discipline. The line discipline filters the mouse data and then passes a generic data packet to the user through normal read system calls or calls the user-specified emulator with the data packet. This line discipline is explained in the *tb*(4) manual page.

For compatibility, the default line discipline may be set using the upper four bits of the minor device number. To get the device interface specified in *mouse*(4), use the discipline MSLINEDISC from <*machineio/mouseio.h*>. Any new software that uses the mouse should set its desired discipline explicitly.

9. FILES INCLUDED WITH THE EMULATOR

The following tables briefly explain the files contained in the emulator package. The tables specify files according to function. Each table states where the files are located and describes what each file contains. Tables with a column marked *User* distinguish between purely kernel files and user/kernel-shared include files. These user include files are needed to access emulator functions.

Emulator Control Files		
/sys/machinecons		
File	User	Description
cons.c	no	Console driver routes requests to appropriate emulator and its input/output device or to the emulator controller. Console driver is also responsible for console message forwarding.
consdefs.h	no	This file contains hardware interface information about system input devices. Emulators as well as device dependent routines use this to interface with each other and the hardware.
consio.h	yes	Defines which displays and screen controls/flags are available. The ioctl commands and structure for screen_control are in this file. This file is indirectly included by screen_conf.h.
consvars.h	no	Emulator control variables are declared here.
bus.c	no	I/O bus control driver, which allows access to the I/O bus on a per-process basis. Window managers that need to get directly at the display from user space should open /dev/bus.
screen_conf.c	no	Where all the displays and emulators are configured for the system. This file also contains the emulator control routines discussed in "Emulator Package Functions", above.
screen_conf.h	yes	Where all the structures, defines, and macros for the emulator package live. This contains all the macros for an emulator to interface with the device-dependent routines as well as the ioctl information for the user to interface with the emulator package.

Emulators		
/sys/machinecons		
File	User	Description
aed.c aeddefs.h	no	AED raw microcode graphic emulator
buf_emul.c	no	Buffering emulator, which saves messages sent to display, then flushes them when the output emulator is changed
ibm_emul.c mono_tcap.h	no	IBM3101 output emulator; takes a considerable subset of IBM3101 commands defined in tcap
kbd_emul.c kbd_emul.h kbde_codes.h	no	A keyboard emulator which allows mapping of key codes to a character stream
std_emul.c std_emul.h	no	Standard output emulator routines, which send raw characters to the display on output. This output emulator is used for displays that perform their own emulations.
x_emul.c	no	X window system input emulator, which queues up keyboard and mouse events into a memory area shared between kernel and user. This emulator also has a variety of ioctls for controlling the locator on a display, as well as performing other X-related functions (e.g. tracking the cursor, etc.).
xio.h	yes	X-dependent structures and defines for kernel and usr process
qevent.h	yes	Event Queue structures and defines used by the X emulator

New/Changed Line Discipline files		
/sys/sys and /sys/h		
File	User	Description
tty_conf.c	no	Line discipline configure file
tty.h	yes	Line discipline structures and defines
tty_tb.c	no	Normal tablet line discipline changed to support system/serial type mouse devices also; also changed for forwarding data packets to input emulator if specified
tbdefs.h	no	Tablet/mouse generic data packet structures and defines
tbioctl.h	yes	Ioctl commands and structures

Low Level Output Display Dependent Files	
/sys/machinecons	
File	Description
aed_tty.h	Macros and defines for interfacing with the glass tty microcode for the AED display
aed_tty_mcode.h	Glass tty microcode for download to the AED display
aedloc.c	AED locator low-level device-dependent routines
aedtty.c	AED glass tty low-level device-dependent routines
apa16loc.c	APA16 locator low-level device-dependent routines
apa16tty.c	APA16 glass tty low-level device-dependent routines
apa16tty.h	APA16 device-dependent structures and define
apa16tty_font.h	APA16 font for glass tty emulation
apa8cloc.c	APA8 color locator low-level device-dependent routines
apa8ctty.c	APA8 color glass tty low-level device-dependent routines
apa8ctty.h	APA8 color device-dependent structures and define
apa8loc.c	APA8 locator low-level device-dependent routines
apa8tty.c	APA8 glass tty low-level device-dependent routines
apa8tty.h	APA8 device-dependent structures and define
apa8tty_font.h	APA8 and APA8 color font for glass tty emulation
apa_fontblt.c	Generic routines for font manipulation on APA displays
apa_structs.h	Generic structures and defines for font manipulation on APA displays
apaaed.h	Structures and defines for dealing with the AED as an APA display
mono.c	Monochrome glass tty low-level device-dependent routines
monocons.h	Monochrome device-dependent structures
monodefs.h	Monochrome device-dependent defines

Low Level Keyboard Device Dependent Routines	
/sys/machinecons	
File	Description
keyboard.c	System keyboard hardware routines
keyboard.h	System keyboard hardware structures and defines
kls.c	Keyboard/mouse/speaker common routines
kls.h	Keyboard/mouse/speaker low level defines

System Mouse Device Driver		
/sys/machineio		
File	User	Description
mouse.c	no	Driver for system mouse
mouseio.h	yes	System mouse structures and defines; also includes ioctl controls/defines for user processes
mousereg.h	no	System mouse driver declarations
speaker.c	no	Speaker driver
speakerio.h	yes	Speaker structures and defines
speakervar.h	no	Internal speaker data structures

The Remote Virtual Disk System

This article is an updated version of several articles written by J. H. Saltzer, J. Van Sciver, L. W. Allen, P. Prindeville, and Michael Greenwald at MIT between 1983 and 1986. The original articles, based on MIT's Project Athena, have been rewritten and include additions and changes for the IBM RT PC and 4.2/RT.

This article describes the Remote Virtual Disk (RVD) system for use with 4.2/RT on the IBM RT PC. It contains the following chapters:

1. **Overview** contains background information on RVD.
2. **RVD Structure** describes the structure of the RVD system.
3. **Installing RVD** describes RVD installation.
4. **RVD Protocol Specification** describes the RVD communications protocol (optional reading).
5. **RVD Control Protocol Specification** describes the RVD remote server maintenance protocol (optional reading).

1. OVERVIEW

The Remote Virtual Disk (RVD) system is a network service that provides a client computer with the appearance of removable-media disk drives and an unlimited number of removable disk packs. The removable disk packs are actually stored in private regions of large disks on an RVD server. When a remote disk pack is "spun up", it appears to most software to be just another disk drive. Although read and write requests are actually accomplished by sending messages across the network to the server, on a local area network the performance of a remote disk pack is only slightly less than that of a local fixed disk.

RVD is a very simple system. Its only addition to the usual list of functions of a hardware disk is remote access. Its design makes little use of operating system features, so it is fairly independent of the operating system. An RVD client may be implemented for any operating system that allows installation of device drivers, and an RVD server may be implemented under any operating system that permits access to either disk partitions or large files. A server that runs under one operating system may be used by a client that runs under another.

1.1. Remote Virtual Disk Packs

A remote virtual disk pack is a portion of a real disk, located on an RVD server. RVD packs are named and allocated by an administrator for the particular RVD server. The name (a character string) and the size (measured in sectors of 512 bytes) are negotiated between the administrator and the prospective user. Once allocated, the space is reserved on the physical disk for the lifetime of the RVD pack.

When a client computer uses ("spins up") an RVD pack, the client specifies one of two modes of access: read-only access or read/write exclusive access. These modes follow the usual rules for read/write compatibility: there may be several simultaneous readers, or exactly one exclusive-mode user of any one virtual disk.

Access to an RVD pack may be protected by passwords, with a separate one for each of the modes of access. Thus one might protect an RVD pack used as a group library by requiring one password (or no password at all) for read access, and a different password for exclusive access. A private RVD pack might use the same password for both modes. It is also possible, by arrangement with the server's administrator, to specify (by internet address) a preferred client that may spin up a password-protected RVD pack without providing the password.

When a new RVD pack is allocated, the first thing one normally does is create an initialized, empty file system on that pack.

1.2. Supporting Tools

Normally one treats a remote disk pack just like any removable storage medium; all standard commands and tools are applicable. In addition, there are a few specialized tools that are useful in managing the remote disk system.

Client management: The RVD client code is packaged as a driver. There are commands that display information and state of the client part of the RVD system.

Server management: The RVD server is designed to be operated from a distance via a network connection. Client commands are available to invoke any remote management operation of the server.

Remote pack management: A high-speed copy command provides a high-performance way of duplicating the contents of one remote pack onto a second one.

1.3. Hazards

RVD is an example of a distributed system, in which failures of the server and of the data communication network can occur independently of failures in the client. This failure-independence can lead to some situations that might not have been anticipated, or that are so rare when using a local disk that they are not handled well, in the writing of the software of the client operating system or applications.

The most common failure is that a packet is lost in the data communication network. The RVD client-server protocol includes a sophisticated request-retry procedure that will immediately and automatically recover from occasional lost packets. It will also automatically recover from short network outages (up to a minute or so,) although some application programs may have timers that get impatient with the delays involved in waiting for a network to recover.

Generally, longer network outages, or crash and restart of the server, are reported as errors back to the invoking file system; whether or not the user is able to recover depends on the application's response to these errors.

If the client crashes and requires rebooting or is powered down while it has RVD packs spun up, it loses its local record of spunup packs, but the server still has a record. There is a general cleanup function, named *rvdflush*(8), that sends a request to a server to spin down all RVD packs associated with this client. It is general good practice to spin down all packs at the end of a client session, and to run *rvdflush* at the beginning of every client session, in case the previous session ended with a crash.

Most operating systems have a file system integrity-checking program. It is usually necessary to use such a program to review, and if necessary, to repair, the contents of an RVD pack following a crash, just as with a local disk. It is also good practice to run such a program just before using any newly spunup pack, especially if that pack may be used by other clients.

As a general rule, all software works correctly with RVD unless it is written to be dependent on hardware parameters of specific physical disks. However, most operating systems have some programs that know too much; those programs must be avoided.

1.4. Network Protocols

The Remote Virtual Disk system uses two network protocols, named RVD and RVDCTL. The client driver and the server communicate with the RVD protocol, to perform spinup, disk read/write, and spindown. The RVD protocol is a transport protocol, using the Internet Protocol (IP) as its base. It is described in detail in Chapter 3 of this article.

The Remote Virtual Disk Control (RVDCTL) protocol supplies a Remote Virtual Disk server with both operating instructions and information about its configuration. An RVD server process comes into existence with no knowledge of the physical configuration of the system in which it is embedded or the logical configuration of the (possibly already existing) virtual disk packs it is to manage. By supplying this information via a network connection instead of from files on the server host, it becomes possible to administer all aspects of server operation remotely. RVDCTL is an application protocol, using the User Datagram Protocol (UDP) as its base. It is described in detail in Chapter 5 of this document.

2. RVD STRUCTURE

This section describes the general structure of the RVD server. The RVD server is organized as a user program that requires little of the underlying operating system apart from access to a disk device. It is started by invoking (as root, or in */etc/rc.local*) the command *rvdsrv*(8), normally run in the background. The server has no configuration description files. Instead, it opens a network listening port (the RVDCTL port) and expects that someone will send its configuration and all operating instructions to it on that port.

2.1. Authorization

The installer of RVD should create a file named */etc/rvd/rvdauth*, owned by root and readable only by root. It contains an unencrypted ASCII authorization password that the server demands on most uses of the RVDCTL connection. If the contents of *rvdauth* are changed, the change takes effect the next time the control request "require_authorization" is sent to the server, or the next time the server is started.

2.2. File Placement

The server program is installed in */usr/ibm/rvdsrv*. There is a set of server-management commands in */usr/ibm*.

For convenience, the directory */etc/rvd* contains the RVD initialization data file, the text of any user message, and RVD operation logs. These files are described in the next two sections.

2.3. Initialization

As mentioned, the RVD server, once started, takes its configuration initialization, as well as operating instructions and also instructions to change its configuration, as a series of operation requests sent to it over a UDP socket. Since all operation requests for the RVD server are ASCII text strings, RVD server initialization is conventionally accomplished by maintaining somewhere on the server an ASCII text file containing the sequence of initialization instructions. A program named *rvdsend*(8) can be used to send the contents of that file over the control connection.

If a user message is posted at the server, it is a good idea to keep a copy of its text in a standard place so that it can be reposted if the server needs to be reinitialized.

It is convenient to divide the initialization instructions into two files, one of which, *rvddb*(5), initializes only the configuration, while the other (*rvdenable*) contains instructions to start the server operating. A typical invocation of RVD at boot time then appears as the following sequence in the file */etc/rvd/rvdstart*:

```
/etc/rvd/rvdsrv -l 11 &              # start server with logging
/etc/rvd/rvdsend /etc/rvd/rvddb      # set up server configuration
/etc/rvd/rvdsend /etc/rvd/rvdenable  # tell server to start work
```

where the file */etc/rvd/rvdenable* contains:

```
operation=allow_spinups
mode=5
operation=require_authorization
```

One reason for dividing the initialization instructions into two files is that there is a configuration-management program, *vddb*(8), that provides a convenient user interface for creating and changing a file that contains the disk-pack configuration. A second is that while the system is in single-user mode, the file *rvdenable* may temporarily be replaced with an alternate file that starts the server in a different way, perhaps by forbidding any but operations use.

2.4. Logging

The RVD server uses the *syslog*(8) facility. All RVD server logging is to the *syslog* identifier LOCAL7. The RVD server uses the conventional *syslog* levels as follows:

ALERT	(0)	serious server problems from which recovery is unlikely
ERROR	(1)	recoverable server errors such as bad disk blocks
INFO	(2)	spinups, spindowns, and name exchanges
	(8)	errors made by clients: bad passwords, attempt to read packs that aren't spun up, etc.
DEBUG	(4)	all read and write requests
	(16)	complete packet level trace of RVD operation

The numbers in the above list are logging classes. Items in class zero are always logged; RVD has a log control system that allows one to turn each of the other classes of logging on or off independently. The command invocation line for the server includes a parameter (the sum of the class numbers) to turn on the initial logging classes. The command *rvdchlog*(8) sends a control protocol request to change the classes of events that are logged.

The *syslog* configuration file (*/etc/syslog.conf*) can direct all logging output from LOCAL7 to an appropriate file, for example, */etc/rvd/rvdlog*. In addition, if there is a system log that is reviewed daily, RVD logging output of levels ALERT and ERROR might appropriately be directed there, too.

Logging all read/write requests or every packet produces a noticeable performance degradation of the server, so it is not recommended for normal operation. Logging spinups and client errors provides information about usage of the RVD service and also often records entries that suggest particular clients are misconfigured or are having some problem. If spinups and spindowns are logged, a busy server can fill 100 Kbytes of log in a day. Thus it is a good idea that a crontab (see *cron*(8)) entry invoke a nightly script to move the RVD log aside and start a new one.

2.5. Remote Management

The RVD server is designed to allow all management to be done remotely. The program that manages the initialization data, named *vddb*, can be run either locally on the server, or elsewhere in the network. Remote management of several servers can be accomplished by setting up a directory on the management host that contains a copy of the *rvddb* initialization file for each server to be managed, named with the network name of the server. When *vddb* is invoked with the name of the server, all configuration changes that the system administrator requests are made to the central *rvddb* initialization file for that server and they are also performed, via the control connection, on the server itself. If the server is set up to reboot and reinitialize itself automatically from a local copy of the *rvddb* initialization file, the system administrator should, when *vddb* has completed, copy the newly modified initialization file to the server.

2.6. Remote Partition Management

The command *savervd*(8) copies virtual disk packs to tape, and *zaprvd* (see *savervd*(8)) does the reverse. If the server does not have a tape drive attached, it is possible to do this operation remotely, by using RVD twice. The basic trick is to set up links in */dev* so that there are two names for every disk partition managed by the RVD server. The server uses one of these names (e.g. *vdsrv1*) for virtual pack assignment; it uses the other name (e.g. *rdvdsrv1*) for a single virtual pack that overlays the entire partition. That overlaying virtual pack can then be spun up on a remote system that has a tape drive available. Once spun up, the raw RVD device that represents the spunup virtual pack can be treated just like a local disk partition on the remote system, and another copy of the RVD server can be operated on that system. A symbolic link to that raw RVD device, but with the same name as the link used for virtual pack assignment on the original system (e.g. *vdsrv1*) allows a copy of the *rvddb* from the first server to be reused at the remote site. This trick allows *savervd* and *zaprvd* to think they are operating on the original server.

3. INSTALLING RVD

This section describes how to install RVD on an IBM RT PC running 4.2/RT. The first part gives a description of installation; the second part is a step-by-step description of installation.

3.1. Description of RVD Installation

As the RVD system follows a client-server model, there are two kinds of installation procedures. Two scripts in */etc/rvd* facilitate these procedures: *rvd.mkserver* for the server machine, and *rvd.mkclient* for the client machine. These scripts are user-friendly front ends for RVD installation procedures.

3.1.1. Installing an RVD Server Machine

Installing an RVD server machine involves the following tasks:

- Creating the required virtual disks
- Configuring the file systems that the RVD disk packs will hold
- Starting the RVD server program

3.1.1.1. Creating Virtual Disks

Creating the disks requires assigning local physical disk drives to hold the required file systems, and configuring a data base file *rvddb* for the RVD software (see *rvddb*(5) and *vddb*(8)). The *rvd.mkserver* script will prompt the user for existing devices, and will create links (named *vdsrv0* through *vdsrv9*) to these devices. These links serve as RVD's interface to the physical disks. During the creation of the *rvddb* data base, the *vddb* program prompts the user for information associated with the partitioning of the chosen disk drives and the allotment of these partitions to RVD disk packs. The program uses the information supplied by the user to create the *rvddb* file, which is in turn used by the RVD server program.

3.1.1.2. Configuring the File Systems

The administrator for the RVD server system will determine what file systems the RVD packs will hold. The decision is not particularly sensitive, except in the case where one mounts the */usr* file system via RVD. Because the */usr* file system is particularly large, it is desirable to mount it via RVD on client machines. However, the */usr* file system contains files and directories which must be local to the client machine. (Generally, any file or directory which must be writable must be local.) When "usr" is specified as a pack name, the *rvd.mkserver* script creates and makes symbolic links to a new file system, */site*, which is set up to contain the necessary local files and directories. Some of these files and directories, such as */usr/spool*, are put in */site* by default; others are put in the *usr* RVD pack by default. The user is prompted to determine the disposition of files and directories in */usr* which are not recognized by *rvd.mkserver*.

Because there will most likely be occasional changes made to the *usr* pack, it will be necessary to keep a backup *usr* pack on the server machine. Making changes to an RVD pack requires that that pack be spun up in the exclusive read/write mode, an operation which cannot be performed on a pack spun up by any other RVD user. In general, when configuring an RVD server machine, one should keep in mind the need for space for a backup copy of any RVD pack that will require changes while in service.

3.1.1.3. Starting the RVD Server Program

Finally, *rvd.mkserver* prompts the user to set the RVD command authorization password, if it has not already been set in the file */etc/rvd/rvdauth*, and the RVD server program *rvdsrv* is started. Full RVD services are available at the completion of the *rvd.mkserver* script.

3.1.2. Installing an RVD Client Machine

Installing an RVD client machine involves the following tasks:

- Creating the *rvdtab* data base (see *rvdtab*(5)) and, perhaps, modifying */etc/fstab*
- Rebooting the client machine to reconfigure and remount its file systems if the */usr* file system is being mounted by RVD.
- Setting up the *rvdusr* file tree if the *rvdusr* pack is to be used

3.1.2.1. Creating the rvdtab Data Base

The first thing the *rvd.mkclient* script does is check for the existence of the *rvdtab* file (nominally at */etc/rvd/rvdtab*). If the *rvdtab* file already exists, *rvd.mkclient* echoes its contents to the user. Then *rvd.mkclient* prompts the user for the information required to build or extend the *rvdtab* file. When the file is completed, its contents are echoed to the user for inspection.

After dealing with the *rvdtab* file, *rvd.mkclient* creates the directories used as mount points for selected RVD packs, e.g. *rvdusr* and *src*. If the *usr* pack is going to be used, the */site* directory is created and */etc/fstab* is modified so that the local */usr* file system will be mounted at */site* upon reboot. The *usr* pack spun up from a server comes with symbolic links to */site* for all files and directories required to be kept in local (non-RVD) storage. These links are set up automatically when the RVD *usr* pack is created using *rvd.mkserver.*

3.1.2.2. Rebooting the Client Machine

Rvd.mkclient gives its user a 30-second warning before calling for a system shutdown in one minute. After shutdown, the system automatically reboots, coming back up with the appropriate RVD client configuration. At this time any packs designated as "default" in the creation of *rvdtab* are spun up (barring network and server problems). If the *usr* pack is being used, it is spun up at this juncture; any network or server problems preventing the spinup of the *usr* pack will cause the reboot process to hang. (This is one good reason to have the *usr* pack available on more than one server machine.)

3.1.2.3. Setting up the rvdusr File Tree

After the client machine is rebooted, the user needs to spin up any packs currently needed but not specified as default packs in the creation of *rvdtab.* If the *rvdusr* pack is being spun up to have parts of the */usr* file system accessed via RVD, the user will want to execute the *rvdusr.config* script. This script goes through all files and directories at the level directly below the root of the */rvdusr* file system (that is, the *rvdusr* pack) one at a time, and tells the user exactly how much local disk space can be saved by accessing that file or directory via RVD. This part of the process takes about 30 minutes. The user is then prompted for the choice between RVD and local storage, and the appropriate action is taken. If the choice is local storage, nothing is done (i.e. the file/directory stays on the local */usr* file system disk). If the choice is to put the file/directory on RVD, it is removed from the local disk and replaced by a symbolic link to the corresponding file or directory in */rvdusr*. The space previously occupied by that file or directory on the local disk is then free for other uses.

3.1.3. Notes on Installing RVD

When a client machine is mounting its */usr* file system via RVD, it is prudent to have the */usr* pack available on more than one server machine. A client machine cannot afford to be without a */usr* file system, since the executable code for many fundamental 4.2/RT utilities resides in the */usr* directory. Thus the loss of access to the server for the *usr* pack, for any reason, is a serious handicap and should be avoided through having multiple servers available with this pack. *Rvd.mkclient* reminds the user of this when creating the *rvdtab* file.

The */usr/src* file system is also a good candidate for installation as an RVD pack. It must be mounted separately from the usr pack, as it is a distinct file system. Note that this file system has the classic characteristics of a good RVD pack candidate: it requires a large amount of disk space, will not be written to by the average user, and will be written to or changed (by anyone) only infrequently. Such file systems can be accessed in read-only mode by many users simultaneously, and rarely require any attention on the part of the system administrator.

Care should be taken if new workstations are to be installed at your site via network connections (see "RVD Installation Steps" below). The source machine for such installations should in general NOT be an RVD server; if it is, the target machine will end up being an identical server machine. (However, this might be useful for servers featuring the *usr* pack.) Also, target machines should be installed over the network before their configuration as RVD clients. This maintains full flexibility in that configuration process.

3.2. RVD Installation Steps

This section describes the steps involved in installing an RVD server machine and an RVD client machine, and the scripts (*/etc/rvd/rvd.mkserver* and */etc/rvd/rvd.mkclient*) used to do this.

3.2.1. Installing an RVD Server Machine

Installing an RVD server machine is done by using the *rvd.mkserver* script, which resides in */etc/rvd*. This script provides an easy and thorough procedure for the installation. This document describes the prompts provided by *rvd.mkserver* and the appropriate user responses.

3.2.1.1. Starting the rvd.mkserver Script

Before using *rvd.mkserver*, you must first *su* to root, because of the major changes being made. Failure to do so will result in the following error message:

you must su to root to run this script

3.2.2. Creating Virtual Disks

Upon successful invocation, *rvd.mkserver* responds with the following:

creating virtual disk drives:
we will create links to existing devices, which
should be drivers for existing local disk drives.

here is a list of currently mounted physical devices:

What follows is the output of the command *mount* which shows the available mounted physical disk drives. These drives are candidates for use in creation of the

RVD virtual disk drives. Note that there may also be unmounted disk drives which could also be used.

There are 10 links, named *vdsrv0* through *vdsrv9*, that *rvd.mkserver* makes available. *Rvd.mkserver* automatically keeps track of the next available link and presents the following prompt, where X is between 0 and 9:

> *next available link is vdsrvX - make link? (y/n)*

A response of y or **yes** (referred to as an "affirmative"response in this chapter) will continue the process of creating the link; all other responses will cause *rvd.mkserver* to move on to creating the *rvddb* data base file (see below).

An affirmative response to the above prompt generates this prompt:

> *which device to link to? (< list of devices > CR)*

where *< list of devices >* is a list of (mounted) candidate devices.

Type your response in the following format:

> **/dev/disk_drive***X*

or

> **disk_drive***X*

or **< CR >**, where *disk_driveX* is an existing *block special* or *character special* file. A carriage return (**< CR >**) will go on to the next available device; an incorrect response will generate the error message:

> *device must be a disk driveX, (your reply) is not*

Rvd.mkserver will loop until it receives a satisfactory reply. Once that reply is received, it responds as follows, where X is again between 0 and 9:

> *linking /dev/vdsrvX to (your reply)*

3.2.2.1. Creating the rvddb Data Base File

Next *rvd.mkserver* will respond:

> *creating rvd data base /etc/rvd/rvddb...*

If there is already a file */etc/rvd/rvddb* in existence, *rvd.mkserver* responds with:

> */etc/rvd/rvddb already exists - it looks like this:*

whereupon the contents of that file are echoed to the terminal. (Note that the file may be empty!)

The *rvddb* file is a data base of RVD virtual disk partitions (see *rvddb*(5)). *Rvd.mkserver* gives the user an opportunity to create or modify *rvddb* with the following prompt:

> *do you wish to modify /etc/rvd/rvddb? (y/n)*

There is a standard RVD utility program *vddb* (see *vddb*(8)) used for creating and modifying *rvddb* which *rvd.mkserver* calls upon an affirmative response, with this message:

> *calling vddb (see vddb(8)).*

As *vddb* is a fairly obscure program to the new user, *rvd.mkserver* offers the user a glimpse of what a typical session with *vddb* might look like, with the prompt:

> *would you like to see an example first? (y/n)*

An affirmative response generates the following:

a typical session might go like this:

```
Ready
> add physical
Physical disk file name: /dev/vdsrv0
Size in 512-byte blocks: 88536    (note: see diskpart(8) for more info on this)
Are you sure (y or n)? y

Ready
> add virtual
Virtual disk name: src
Description: /usr/src file system
Read-only password:
Exclusive password: src_password
Shared password:
Disk size in 512-byte blocks: 88536
Allowable modes: 5
Owning host ( < CR > for none):
Physical disk name ( < CR > for any): /dev/vdsrv0
Are you sure (y or n)? y

Ready
> quit
```

hit return key to begin vddb session:

Pressing **< Enter >** causes *rvd.mkserver* to invoke *vddb*. Invoking *vddb* requires the RVD *command authorization password*, which *rvd.mkserver* will prompt you for. Note that this password will be the null string (equivalent to a carriage return) if it has not been previously set, thus the prompt for a password may be ignored. If it has been set it will be found in */etc/rvd/rvdauth*, where it is readable by *root*.

3.2.2.2. Configuration of a usr RVD Pack

After finishing the session with *vddb*, *rvd.mkserver* will search the */etc/rvd/rvddb* data base file to see if a pack named *usr* is being created. If it finds mention of such a pack in */etc/rvd/rvddb*, a complex series of actions is initiated. If your installation does not include a *usr* pack, you may skip this section.

Rvd.mkserver signals that it has found that a *usr* pack is being created with this message:

/usr will be an rvd disk pack, must make arrangements...

The *usr* RVD pack is used to hold the */usr* file system. The */usr* file system is complex, and contains several directories which generally must be local (i.e., non-RVD) because they require write permission. Thus *rvd.mkserver* proceeds to create a directory */site* in the *root* (or /) file system to hold these local directories. First *rvd.mkserver* checks to see if there is enough free space on the *root* file system for the directories which it knows that */site* must hold. If *rvd.mkserver* finds that there is not enough space it will display the following:

```
there is not enough space on / for /site
we need X kilobytes for /site
there are only Y kilobytes available on /
exiting
```

If this happens, *rvd.mkserver* exits and the system administrator must reconfigure the local disks to provide the necessary space if he or she wishes to have the *usr* RVD pack.[1]

If *rvd.mkserver* finds that there is sufficient space for */site*, it will respond with the following:

creation of /site will require X kilobytes in / file system
there are Y kilobytes free in / file system
do you wish to proceed with creating /site? (y/n)

An affirmative response generates:

proceeding...

Any other response causes *rvd.mkserver* to exit with:

exiting

The above choice is given because the figures given by *rvd.mkserver* may indicate that the creation of */site* will leave an unacceptably small amount of free space on the *root* file system. Thus the system administrator may wish to reconfigure the local disks to allow more space on the *root* file system.

If the user has chosen to proceed with the creation of */site*, *rvd.mkserver* displays the following:

creating /site on / file system to store local /usr files

At this juncture *rvd.mkserver* creates and/or set the correct access modes on */site*, then proceeds to move those directories it knows in advance must be local from */usr* to */site* with this message:

moving (adm guest msgs preserve spool tmp) from /usr to /site:

Before actually moving a directory, *rvd.mkserver* will check to see if that directory has already been linked to */site*. If it is, this message will appear:

/usr/(directory) is already linked to /site/(directory)

where (directory) is the directory in question. Also, *rvd.mkserver* will check to see if a directory of that name already exits in */site*. If it finds one, this appears:

/site/(directory) already exists - best check it. we will proceed.

and no action is taken. If neither of the above errors occurs the move and link are made and the directory name is echoed. When finished with all directories, *rvd.mkserver* outputs:

...done.

Next *rvd.mkserver* moves and links */usr/lib/crontab* to */site* with the message:

moving and linking /usr/lib/crontab to /site/lib/crontab

If */usr/lib/crontab* is already linked to */site*, this message will appear:

/usr/lib/crontab already linked to /site/lib/crontab

Rvd.mkserver has a list of directories it expects to find in all */usr* file systems, and automatically disposes as RVD or local. It also checks for directories not in that list and prompts the user to determine their disposal. That process begins with this message:

[1]Note that there may be even more space required for */site* than is indicated by *rvd.mkserver* at this step. The estimate given is based only on the space required by the directories that *rvd.mkserver* knows *a priori* must be local. There may be other directories peculiar to your site that must also be local, and thus accommodated on */site*.

looking at other files & directories in /usr to make local or remote
default is local (not on rvd pack)

One by one *rvd.mkserver* gets the size of each directory, reviews the space left on the *root* file system, and responds with:

looking at /usr/(directory)...
there are X kilobytes left on /site
/usr/(directory) requires Y kilobytes
put /usr/(directory) on usr rvd pack? (default is move to /site) (y/n)

An affirmative response means that the directory will stay on the */usr* file system rather than being moved to */site.* Such directories will then be available to RVD clients on the *usr* RVD pack. Each client that uses the *usr* pack will save the "Y" kilobytes of local disk space required by that directory, but that client will generally not be able to write to that directory. Thus *write permission* and *space savings* are the two factors that should be weighed in deciding this response. An affirmative response generates the reply:

/usr/(directory) will be on usr rvd pack

A non-affirmative response means that the directory should be local. Thus *rvd.mkserver* attempts to move the directory to */site* and make a link from */usr* to */site.* If the directory is already a link to */site rvd.mkserver* displays:

/usr/(directory) already linked to /site/(directory)

If a directory of the same name already exists on */site*, this appears:

/site/(directory) already exists - best check it. we will proceed.

and *rvd.mkserver* proceeds without taking any action.

Otherwise, if all goes well, this appears:

moving and linking /usr/(directory) to /site/(directory)

3.2.2.3. Setting the RVD Command Password

The RVD system requires a *command authorization password* to accompany all network commands for remote maintenance of the RVD server. If this password is not already set (in the file */etc/rvd/rvdauth*) *rvd.mkserver* prompts the user to set that password:

setting rvd command authorization password
enter new password:

After the user has entered the password, *rvd.mkserver* responds:

installing new password in /etc/rvd/rvdauth... done.

3.2.2.4. Setting RVD Operation Modes

Another requirement for RVD operation is a file */etc/rvd/rvdenable* which contains information on the operational modes of the RVD server. If it doesn't already exist, *rvd.mkserver* creates that file and displays the following:

creating /etc/rvd/rvdenable file... done.

Note that this step requires no input from the user.

3.2.2.5. Starting the RVD Server Daemon

Finally, *rvd.mkserver* starts the RVD server daemon after displaying this message:

starting rvd server program

and prompting for the *command authorization password* (which again will be null if not previously set in */etc/rvd/rvdauth*). If the RVD server daemon was already running at this point *rvd.mkserver* displays:

rvd server program is already running - no further actions will be taken

3.2.2.6. Setting the RVD System Message

An RVD server has an RVD system message option. This message is used to communicate with RVD clients about new packs, changes to existing packs, etc. *Rvd.mkserver* calls *rvdsetm*(8) to allow the user to set this RVD system message. *Rvdsetm*(8) prompts for the message, which you type in and terminate with a <CTRL-D>. Then *rvdsetm*(8) will prompt for the RVD *command authorization password*, which may again be the null string.

The installation of the RVD server is now complete. *Rvd.mkserver* exits with a status of 0, and all RVD services are now available on this server machine.

3.2.3. Installing an RVD Client Machine

Installing an RVD client machine is done using the *rvd.mkclient* script, which resides in */etc/rvd.* This script provides an easy and thorough procedure for the installation. This section describes the prompts provided by *rvd.mkclient* and the appropriate user responses.

3.2.3.1. Starting the rvd.mkclient Script

Before starting you must *su* to root, because of the major changes being made. Failure to do so will result in the following error message:

you must su to root to run this script

Upon successful invocation, *rvd.mkclient* will first check to see if the RVD devices exist in */dev*. If not, *rvd.mkclient* will create them with a call to *MAKEDEV* after presenting this message:

making rvd devices...

If *rvd.mkclient* makes the RVD devices, it will next show the user what devices it created:

here is a list of currently available rvd devices:

```
crw-rw-rw-  1 root   17,  0 Aug  7 11:47 /dev/rvd0a
crw-rw-rw-  1 root   17,  1 Aug  7 11:21 /dev/rvd1a
crw-rw-rw-  1 root   17,  2 Aug  7 11:21 /dev/rvd2a
crw-rw-rw-  1 root   17,  3 Aug  7 11:21 /dev/rvd3a
crw-rw-rw-  1 root   17,  4 Aug  7 11:21 /dev/rvd4a
crw-rw-rw-  1 root   17,  5 Aug  7 11:21 /dev/rvd5a
crw-rw-rw-  1 root   17,  6 Aug  7 11:21 /dev/rvd6a
crw-rw-rw-  1 root   17,  7 Aug  7 11:21 /dev/rvd7a
brw-rw-rw-  1 root    6,  0 Aug  7 11:21 /dev/vd0a
brw-rw-rw-  1 root    6,  1 Aug  7 11:21 /dev/vd1a
brw-rw-rw-  1 root    6,  2 Aug  7 11:21 /dev/vd2a
brw-rw-rw-  1 root    6,  3 Aug  7 11:21 /dev/vd3a
brw-rw-rw-  1 root    6,  4 Aug  7 11:21 /dev/vd4a
brw-rw-rw-  1 root    6,  5 Aug  7 11:21 /dev/vd5a
brw-rw-rw-  1 root    6,  6 Aug  7 11:21 /dev/vd6a
brw-rw-rw-  1 root    6,  7 Aug  7 11:21 /dev/vd7a
```

Note that this process requires no input from the user.

3.2.3.2. Creating rvdtab

The next step *rvd.mkclient* takes is to give the user a chance to create or modify the */etc/rvd/rvdtab* file (see *rvdtab*(5)). *Rvdtab* stores information concerning the RVD packs that this client uses; *rvd.mkclient* will allow the user to easily build or extend this file. *Rvd.mkclient* announces this phase of the installation procedure with this message:

creating/modifying /etc/rvd/rvdtab...

First *rvd.mkclient* checks for the existence of */etc/rvd/rvdtab.* If the file is found, *rvd.mkclient* gives this message:

/etc/rvd/rvdtab already exists - it looks like this:

After this the contents of */etc/rvd/rvdtab* are echoed to the user. (Note that the file may be empty!)

Then *rvd.mkclient* gives the user the chance to add new RVD packs to the *rvdtab* file with this prompt:

enter a new pack into database? (y/n)

An affirmative response to this query causes *rvd.mkclient* to prompt the user for all the fields required for an *rvdtab* entry. This process will be repeated until *rvd.mkclient* receives a non-affirmative response from the user. If *rvd.mkclient* receives such a non-affirmative response when there was previously no *rvdtab* file and no packs have been added to *rvdtab*, this message appears:

you must create an rvdtab file to have an rvd client machine
exiting

and the script exits.

3.2.3.2.1. Pack Name

First *rvd.mkclient* prompts for the RVD pack name. This name must be the same as the pack name on the server machine, typically something such as *usr* or *src.* (Note that the pack *name* is not necessarily the same as the pack's *pathname*; for example, the pack named *usr* is mounted as */usr.*) *Rvd.mkclient* asks for the pack name:

what will the pack's name be?

Type the pack name after this prompt.

3.2.3.2.2. RVD Pack Mounting Status

Next *rvd.mkclient* asks the user what the mounting status of this pack is to be. The pack may be specified as an essential pack which must be mounted for client operation (as with the *usr* pack), in which case there will be several attempts to spin it up at boot time; as a "default" pack for which one attempt to spin up should be made at boot time; or as a pack which is not to be mounted automatically at boot time. *Rvd.mkclient* asks for this information with this message:

is this pack to be mounted by default (d) or is it
absolutely-must-be-mounted (a)? (default is no mount)
enter a, d, or <Enter>:

If *rvd.mkclient* does not recognize the user's response, it goes back to the beginning of the add-new-pack loop with this error message:

flag (response) is not known; starting over

where (response) is the user's input.

If the name of this pack is *usr*, *rvd.mkclient* does not prompt the user for input, but rather responds with:

usr pack is being made a must-be-mounted pack

This done because the *usr* pack is assumed to contain the */usr* file system, without which the functionality of a UNIX operating system machine is severely compromised.

3.2.3.2.3. RVD Pack Spinup Modes

Now *rvd.mkclient* needs to know what spinup mode(s) to allow for this pack. There are three options: read-only, exclusive read/write, or both. *Rvd.mkclient* prompts the user with:

what spinup mode, read-only (r), exclusive read/write (x),
or both (rx), do you want? (default is read-only)

If *rvd.mkclient* gets either a null string or an unrecognized response it responds with:

using default (read-only) mode

and will make this pack a read-only pack in *rvdtab.*

3.2.3.2.4. Server Machine for RVD Pack

Next *rvd.mkclient* needs to know upon which server this pack resides. Since having the *usr* pack spun up is essential to the normal operation of an RVD client which uses that pack, *rvd.mkclient* reminds its user of the importance of having the *usr* pack available on more than one server for reliability's sake with this message (given only if the pack name is *usr*):

NOTE: we suggest that usr pack be available on more than one server

(Note that only ONE server machine may be specified per entry in *rvdtab.*)

Rvd.mkclient then prompts with:

on what server(s) does this pack reside?

If the response received is null (i.e., <Enter>) *rvd.mkclient* gives this message:

this is not an optional field...

and prompts for the server machine again.

3.2.3.2.5. RVD Drive Number

There are ten drives available for RVD packs. A particular drive number may be specified, but generally the default (<Enter>) is used:

what drive number (0-9)? (default is don't care)

If a null or unrecognized response is received, *rvd.mkclient* responds with:

using default any-number-available drive number

and uses the default (any drive number) for this pack.

3.2.3.2.6. Mount Point

Next comes the issue of where this pack will be mounted. This is an optional field, with defaults for some pack names:

where do you want to mount this file system? (default is /usr for *usr pack, /rvdusr for rvdusr, and /usr/src for src)*

Here you should enter the *pathname* of the directory where this RVD pack is to be mounted when spun up.

3.2.3.2.7. RVD Pack Password

There may be passwords required for access to an RVD pack. If these passwords exist and are known, they may be entered at this prompt:

enter the password for this pack, if any:

Null responses to the above prompt are typical.

3.2.3.2.8. Comments in rvdtab

There may be (single line) comments associated with each pack in the *rvdtab* file. This comment is optional, and can be typed in response to this prompt:

any comment to associate with this pack in the data base entry?

3.2.3.2.9. Finishing the Loop

At this point *rvd.mkclient* will ask again if the user wishes to enter another pack in *rvdtab.* The user may enter as many or as few packs as desired. When a non-affirmative response to the offer to enter a new pack causes *rvd.mkclient* to exit that loop and the user has altered *rvdtab* by adding one or more packs, this message appears:

/etc/rvd/rvdtab now looks like this:

and the contents of the *rvdtab* file are echoed to the user.

3.2.3.3. Creating Mount Points

It is the user's responsibility to provide the mount directories for an RVD pack (see *mkdir*(8)). In the case of the *usr* pack, *rvd.mkclient* assumes that this mount point already exists. In the case that RVD packs *rvdusr* and/or *src* have been added to *rvdtab*, and no other mount point was specified by the user when prompted, *rvd.mkclient* will respond:

making /rvdusr as remote mount point

or

making /usr/src as remote mount point

or both. *Rvd.mkclient* creates the directories mentioned, if these messages appear.

3.2.3.4. Rebooting the Client Machine for usr Pack

If the *usr* RVD pack is being used, the file systems on the client machine must be reshuffled to move the */usr* file system to */site*, thus clearing the way for the *usr* pack to be mounted as the */usr* file system. *Rvd.mkclient* will do this automatically, after giving the user this warning:

starting major changes in 30 seconds; interrupt now
if you don't want your /usr mounted via rvd

At this juncture, *rvd.mkclient* will edit */etc/fstab* to have the */usr* file system remounted as */site*, if not interrupted by the user with <CTRL-C>. Then *rvd.mkclient* gives a second warning:

You have 30 seconds to cancel an automatic system reboot

If not interrupted in this time, *rvd.mkclient* will call for a system shutdown in one minute:

shutting down in 1 minute...

When the machine has been successfully rebooted it will have the *usr* RVD pack in place and spun up. Note that the */usr* file system has been moved to */site* at this point and may have redundant files (to be found on the *usr* RVD pack) which may be culled to save disk space.

3.2.4. Configuring an rvdusr Pack on a Client Machine

The *rvdusr* RVD pack is used when a client does not want to rely upon a server machine for its entire */usr* file system, but would like to access selected directories in */usr* via RVD in order to save space on local disks. When a client machine decides to use the *rvdusr* pack, the system administrator needs to decide which files and directories in */usr* to delete from local storage and link the *rvdusr* pack (which is mounted as */rvdusr* by default). The *rvdusr.config* script found in */etc/rvd* is designed to facilitate this process.

3.2.4.1.1. Invoking rvdusr.config

The user of *rvdusr* may need to have special permissions in order to remove the necessary directories from */usr*. Check the ownership and permissions in */usr* before running this script, or simply *su* to root before using the script.

3.2.4.1.2. Running rvdusr.config

Rvdusr checks to see what directories are available in */rvdusr* and, one by one, offers you the choice of removing the corresponding directory from the local */usr* file system and replacing it with a link to *rvdusr*. After this is done, all accesses to that directory are made via RVD and the space taken up by that directory on the local */usr* file system is freed for other uses. Note that RVD access to files and directories generally *read-only*; this should be taken into account when deciding about whether to keep a directory locally or access it via RVD.

For each directory in */rvdusr rvdusr.config* will first display this message:

looking at /usr/(directory)...

where (directory) is the next directory, alphabetically. At this point *rvdusr.config* gets the amount of space that that directory occupies on the local disk, and comes back with this:

you can save X blocks by putting (directory) on rvd
put (directory) on rvd? (keep local is default) (y/n)

An affirmative response causes *rvdusr.config* to remove (directory) from the */usr* file system and to replace it with a symbolic link to */rvdusr* with this message:

linking /usr/(directory) to /rvdusr/(directory)...

and after a brief pause during which the changes are made:

done

If the response is not affirmative, *rvdusr.config* uses the default action, which is to leave things as they are:

keeping /usr/(directory) on local disk drive

After all directories in */rvdusr* are covered, *rvdusr.config* exits with a status of 0.

3.2.4.1.3. Recovering RVD Directories to Local Storage

Should the user(s) of the RVD client machine ever wish to return any directory of */rvdusr* to local storage in */usr*, the symbolic link to */rvdusr* can be removed and the directory copied from */rvdusr* back to */usr*.

4. RVD PROTOCOL SPECIFICATION

This section is optional reading.

4.1. Introduction

4.1.1. Motivation

The Remote Virtual Disk (RVD) Protocol provides the ability to dynamically attach arbitrary disks of different sizes to arbitrary computers. It is especially useful when the computers are physically remote or when user intervention is impractical (e.g. when local disks are non-removable, removable packs are expensive, or a wide variety of disk sizes is desired). The RVD Protocol allows either exclusive or shared use of remote devices. The latter mode is useful as a low overhead means of sharing read-only data among physically remote machines.

4.1.2. Scope

The Remote Virtual Disk Protocol simply allows network access to additional disk drives. The protocol does not provide any services beyond those provided by existing disk drives. Specifically, the RVD Protocol:

- does not implement sharing, or any form of object storage or naming mechanism, other than that found on any disk drive.
- does not guarantee reliable writes to the disk.
- does not attempt to resolve architectural byte ordering differences among machines.

This design of this protocol has been concerned primarily with simplicity of implementation, ease of use, and performance.

4.1.3. Use

The RVD protocol is layered on top of the DoD IP protocol and an IP protocol number of 66 decimal (102 octal) has been assigned. This protocol corresponds to level three of the ISO networking standard.

The RVD protocol allows a client machine to communicate with a remote server machine's disk drives as if they were local drives. On the client side, the protocol is used by the I/O subsystem software, typically a device driver, to communicate with the remote drive. All client software would then use the client device driver to treat the remote drive as if it were just another local device. On the server side, either an application process or the addition of operating system support could be used to make the connection between protocol requests and local disk requests. The level at which the server is implemented is not part of this specification.

4.2. Specification

The RVD Protocol is used in a client/server scenario. The client makes requests and the server responds. The protocol defines four request/response pairs and an error response. The pairs are (request/response):

SPIN-UP/SPIN-ACK:
: The client requests use of one of the server's drives, and the server acknowledges the client's spin-up request.

SPIN-DOWN/SPIN-DOWN-ACK:
: The client disconnects from one of the server's drives, and the server acknowledges the disconnection.

READ/BLOCK:
: The client requests one or more blocks from a spun-up device, and the server responds with the requested data blocks.

WRITE/WRITE-ACK:
: The client writes blocks to a spun-up device, and the server acknowledges one or more write requests.

ERROR:
: The server reports a protocol error while processing a client request; e.g., an error response would be used if the client had omitted a required field in an RVD packet. This response is not used for operation errors; e.g., if a failure occurs when writing to a physical disk then this failure is reported in the status word of the WRITE-ACK response.

The request/response dialog is conducted by exchanging packets between the client and server. All RVD packets consist of a standard header followed by data or parameters specific to the packet type. Descriptions of the RVD packet header and of each packet type follow.

4.2.1. RVD Packet Format

Generalized Format of RVD Packet

<Byte 0>	<Byte 1>	<Byte 2>	<Byte 3>
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Specific Parameters			

- Packet Type

  This byte specifies one of nine RVD packet types:

  SPIN-UP
  SPIN-ACK
  SPIN-DOWN
  SPIN-DOWN-ACK
  READ
  BLOCK
  WRITE
  WRITE-ACK
  ERROR

- Padding

  These two bytes are unused by most of the packet types. SPIN-UP and ERROR do allocate the first byte of this field for their own purposes.

- RVD Version

  The client and server set this field to the version number of the RVD Protocol being used. If either party discovers a mismatch between version numbers then an error occurs. If the server makes the discovery it returns a protocol ERROR packet to the client. This packet will describe the mismatch error.

- Drive

  An index that represents the drive number on the client machine. It is used by the client to specify a virtual disk drive on the server. This index is an integer between zero and (2**32)-1 and is encoded as a 32 bit binary integer. Drive numbers are unique on a given client but there is no correlation between drive numbers on different clients. The server uniquely identifies a virtual disk by the client-drive pair. The server returns the given drive number in its response to the client.

- Nonce

  A 32-bit unique identifier. The nonce is an unsigned integer between zero and (2**32)-1, encoded as a 32-bit binary integer. Since the nonce is a fixed range number it will be unique only over a fixed period of time. It is assumed to be unique for an interval of time that is several times the lifetime of a single packet. The nonce is used to identify a request/response dialog between the client and server. As such, the client inserts a nonce value into its request packet and the server will insert the same nonce into the appropriate response packet.

- Index

  The index is a hint to the server on how to find connection specific information. (A connection is a virtual drive/client drive pairing.) The index has no predefined meaning and the server may use it as any manner of hint desired. The only client request which does not specify an index is SPIN-UP. The responding SPIN-ACK packet will contain, among other things, the server Index, representing the connection that was created by the SPIN-UP request. It is up to the client to return this index with every packet that goes out to this virtual drive.

  If the user ever submits an incorrect Index the server will still find the connection information. It will then send an ERROR packet specifying the correct index. The server would still process the request normally.

- Checksum

  A 32-bit *checksum* of the packet. The *checksum* is the only assurance of reliable data transfer. It is assumed that if the *checksum* is correct then the data is the same as on the disk.

  The *checksum* is computed by adding together all the 32-bit words in the packet. The *checksum* field is considered to be zero for this computation. If the packet does not end on a 32 bit boundary, then the *checksum* computation assumes that the packet is padded out by zeros. The low order 32 bits are then used as the checksum. (The 32 bit sum is taken modulo 2**32.) *Checksum* is computed in Vax byte ordering.

- Specific Parameters

  See the descriptions of the packet types for additional parameters.

4.2.2. Packet Format: SPIN-UP

Format of SPIN-UP Packet

<Byte 0>	<Byte 1>	<Byte 2>	<Byte 3>
Packet Type	Mode	Padding	RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Pack Name			
Capability			
Padding			Blocking Factor

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.2.1. Field Definitions for Packet Type: SPIN-UP

- Mode

  This byte describes the access mode of the virtual drive. Once a virtual drive is spun-up in a particular mode any other client who wants to use that same drive (even from another machine) must open it in the same mode. The mode is one of three values:

  - READ-ONLY gives the client read-only access to the drive. Any other client can read the drive as long as they have also spun it up in READ-ONLY mode.
  - SHARED allows read/write access to the drive by more than one user.
  - EXCLUSIVE gives the client read and write access to the disk, but locks the disk so that no other client can access the disk while it is spun up.

- Index

  this is the only packet type that does not specify an index. The server will return an appropriate index in the SPIN-ACK response packet. This value must then be included in all future client request packets.

- Pack Name

  An ASCII string representing the name of the virtual disk pack that a client wishes to associate with the specified local disk drive. The pack name field is a fixed length string of 32 characters. Each of these characters is represented as an 8-bit byte. The string is null terminated unless the name length is greater than or equal to 32 characters. In that case, the string is truncated to the 32-byte pack name field size.

- Capability

  This field is, like the pack name, a maximum 32-character, null terminated, ASCII string. There are separate capabilities for each drive in each of the spin-up modes. If the mode were, for example, READ-ONLY, then the client would fill in the capability field with the READ-ONLY capability string for this drive.

- Blocking Factor

  This is the read blocking factor, the maximum number of blocks the client can read at one time in a single packet. More exactly, it is the maximum number of 512 byte data blocks that the client will accept in a BLOCK packet type. If the blocking factor is greater than the maximum number of blocks the server can send it will be modified in the SPIN-ACK response packet.

4.2.2.2. Operation

Spinning up a disk establishes a connection between the server's virtual drive and the client's local drive. For example, if the client MYMACHINE wishes to spin up the remote virtual disk "Foo" as his local drive 3, then he sends a SPIN-UP packet to the server. He fills in Packet Type with SPIN-UP and Mode with a valid mode. He fills in Drive with the integer 3. He also supplies the capability for that mode and places the string "Foo" in the Pack Name field.

Upon receipt of the SPIN-UP packet, the server would attempt to fulfill the request. If everything is correct (the drive exists, the capability is correct and so on), the server associates virtual disk "Foo" with drive 3 from client MYMACHINE. From now on, any reference from client MYMACHINE to drive 3 will refer to virtual disk "Foo". The server responds with a SPIN-ACK packet back to the client.

If the server detects an error, it will reply with an ERROR packet. This ERROR packet can be caused by many different classes of errors. First, "real" disk errors; for example, the physical disk containing "Foo" is trashed, or any error that might occur when accessing a physical disk. This is different than the typical case of a computer connected to a physical disk. When accessing a local drive, the error would not be detected until an operation was performed on the drive.

Another type of error is invalid argument values such as a non-existent virtual drive, a bad password, or a bad *checksum.* There can also be inconsistency errors: a disk is already spun up as drive 3, or another client has disk "Foo" spun-up in a conflicting mode. All of these errors will be reported via the ERROR reply.

4.2.3. Packet Format: SPIN-ACK

Format of SPIN-ACK Packet

< Byte 0 >	< Byte 1 >	< Byte 2 >	< Byte 3 >
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Number of Blocks on Drive			
Burst		Queue Length	
Padding			Blocking Factor

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.3.1. Field Definitions for Packet Type: SPIN-ACK

- Index

  The client must save the server's index for use in all future transmissions.

- Number of Blocks

  The server returns the size of the drive in 512 byte blocks. The client should not send any read or write requests for blocks outside of the drive boundaries. If the client does attempt an out of bounds request the server will inform him using the status word in the BLOCK or WRITE-ACK reply packet. The ERROR packet is primarily used for protocol errors.

- Burst

  This is a 2-byte integer that represents the maximum number of packets the server will handle in a single transmission. This value and the blocking factor value are then used by the client when partitioning read and write requests.

- Queue Length

  This is the maximum number of outstanding requests the server will handle for a virtual drive at any one time. This value is different than burst size. The client can send multiple transmissions of burst size packets until the number of packets equals queue length. The client has then saturated the server for this drive and must wait for the server's response.

- Blocking Factor

  This is the read blocking factor, the maximum number of blocks the client can read at one time in a single packet. The value of this field is not necessarily the same value transmitted by the client in the SPIN_UP packet. If the transmitted blocking factor is greater than the maximum number of blocks the server can send, it will be modified in the SPIN-ACK response packet.

4.2.3.2. Operation

The server sends a SPIN-ACK packet in response to a valid SPIN-UP packet. This indicates to the client that the connection request for the server's virtual drive to the client's local drive was successful.

In addition to returning the size of the drive and other connection details, the server provides an index value. Although it is recommended that the client use this index value in all future requests, the server can operate with incorrect Index values. If the server receives a bad index, it will send an ERROR packet of BAD-INDEX type to the user, but the operation will still occur correctly. The most likely cause of a bad index would be the client crashing then attempting to reuse the connection. In all probability the index would be lost, but the BAD-INDEX packet would correct that.

An example helps in explaining the difference between burst size, queue length, and blocking factor. Suppose the client must read forty blocks and that the SPIN-ACK response has reported a blocking factor of two, a burst size of five, and a queue length of ten. The blocking factor limits each read request to two blocks. Thus the client must transmit two sets of five read request packets for the first twenty blocks, wait for the server to respond with the data, then transmit the next two sets of five requests.

4.2.4. Packet Format: SPIN-DOWN

Format of SPIN-DOWN Packet

< Byte 0 >	< Byte 1 >	< Byte 2 >	< Byte 3 >
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Capability			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.4.1. Field Definitions for Packet Type: SPIN-DOWN

- Index

  Should be the index returned by the server for this drive in the SPIN-ACK packet.

4.2.4.2. Operation

If a user wishes to spin down the virtual disk in his local drive 3, then he simply sends a SPIN-DOWN packet to the server. He fills in Packet Type with SPIN-DOWN, and he fills in drive with the 32 bit integer 3. Upon receipt of the SPIN-DOWN packet, the server would attempt to terminate the connection between the client, local drive 3, and the virtual drive. If this worked correctly, the server sends a SPIN-DOWN-ACK back to the client. If the drive was not spun up, or there were some other error, then the server replies with an ERROR packet. It is not polite for a user to consider his disk spun down until he receives the SPIN-DOWN-ACK from the server.

4.2.5. Packet Format: SPIN-DOWN-ACK

Format of SPIN-DOWN-ACK Packet

< Byte 0 >	< Byte 1 >	< Byte 2 >	< Byte 3 >
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.5.1. Field Definitions Packet Type: SPIN-DOWN-ACK

4.2.5.2. Operation

This is the success acknowledgment to a client's spin-down request. Drive, nonce, and index have the same values as those specified by the client.

4.2.6. Packet Format: READ

Format of READ Packet

<Byte 0>	<Byte 1>	<Byte 2>	<Byte 3>
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Starting Block Number			
Block Count			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.6.1. Field Definitions Packet Type: READ

- Starting Block

  The number of the first block that the client wishes to read. This is an absolute offset from the beginning of the virtual drive. This number is a 32 bit integer with a range of zero to (2**32)-1.

- Block Count

  The number of blocks that the client wishes to read.

4.2.6.2. Operation

To read data from a local drive, the client sends a READ packet with the appropriate values. (This drive must have been associated with a virtual disk through a previous spin-up request.) The Drive field is filled with the drive number; the nonce, index, and *checksum* fields are initialized appropriately; and the desired block offset and number of blocks are written into the packet. Upon receiving the packet, the server looks up the connection between the client's local drive and the server's virtual drive.

If the connection exists, the server then tries to send the requested data to the client. If the read request is within the bounds of the disk and the physical read is successful, the server responds with a BLOCK packet. A BLOCK packet contains a block of data, a block number, and a 32 bit status word.

If the server detects an error, it still sends a BLOCK packet, although it has a nonzero status word. Errors can be the result of physical errors on the disk or any of the assorted things that can go wrong on a disk. If the data in the packet is valid, then the invalid-data field in the status word must be zero. The count field in the status word is set to the number of times the server attempted to read the block *before* it was successful. (I.e., if the first try succeeded, the count is zero, if the second try succeeded, count is one, and so on.) If the count exceeds the size of the field, then count is set to the maximum in the field. If the Start Block was invalid, then the bad block address field in the status word will be set. If a multiple block read extends beyond the drive boundaries, then only the in-bounds disk blocks will be returned and the bad-block address field in the status word will be set.

The only time the server sends an ERROR packet in response to a READ is in the case of a malformed READ packet. Protocol errors cause ERROR packets and disk errors cause non-zero status word. Typically ERROR packets will be sent for invalid *checksums*, bad index, or a zero in the Block Count field.

READs can timeout. This protocol does not guarantee delivery of packets. It is assumed that most packets will reach their intended destination, but there are no guarantees. It is up to clients to handle READ timeouts the same way they would treat physical disk timeouts.

4.2.7. Packet Format: BLOCK

Format of BLOCK Packet

< Byte 0 >	< Byte 1 >	< Byte 2 >	< Byte 3 >
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Block Number			
Drive Status			
Data			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.7.1. Field Definitions Packet Type: BLOCK

- Block Number

  This identifies this block in the virtual disk.
- Drive Status

  Drive Status is a 32-bit status word describing the status of the virtual disk. Disk errors are reported through this word. Protocol errors are reported with an ERROR packet.
- Data

  This is data read from the virtual disk. There are (512 x blocking_factor) bytes of data in this field. The *checksum* guarantees reliable data transmission. (The protocol cannot guarantee the accuracy of the disk to protocol data transmission.)

4.2.7.2. Operation

The BLOCK packet is the response to the READ request. It includes the data requested and enough information to allow the client to determine which request this is in response to. (Note that there can be many outstanding READ requests, even in the case where a drive's access is restricted to a single outstanding request. A client can always have requests out to more than one drive.)

In case of an error, the server fills in the appropriate bits in the status. The client must check the data-valid field in the status as the data may be valid even in the case of a non-zero status.

4.2.8. Packet Format: WRITE

Format of WRITE Packet

< Byte 0 >	< Byte 1 >	< Byte 2 >	< Byte 3 >
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Block Number			
Total Blocks in Request			
Index of this Block in Request			
Data			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.8.1. Field Definitions for Packet Type: WRITE

- Block Number

  This is the starting block number for the write. It is an integer that represents the absolute block offset from the beginning of the virtual drive for this data block.

- Total Blocks

  This is the total number of blocks in a sequence of write requests.

- Block Index

  This is the block number of this request.

- Data

  This is the data that is to be written to the virtual drive starting at the specified block number. The data bytes are ordered in the packet exactly the way they are ordered on disk. The bits in a byte are ordered in accordance with the IP specification. The size of the data field is determined by subtracting the size of the header fields from the total size of the packet. The packet size is given by the IP protocol.

4.2.8.2. Operation

The WRITE packet type has been designed to be sent as a burst of packets. If a client wishes to write data to the virtual disk, it creates a sequence of packets, each with the same Total Blocks value but incrementing the Block Number and Block Index. The data is then copied to the data fields of these sequential packets in 1024/512 byte chunks.

Upon receipt of this burst of WRITE packets the server orders the packets, copies the data to a single contiguous buffer, and does the write as one operation. The server sends a WRITEACK if the write was a success, but only for the first packet in the burst.

The WRITEACK includes a 32-bit status word. In the case of a disk error, a WRITEACK with a non-zero status word will be returned. Generally these will be the same type of errors as on a READ request. Additionally, if you try to write to a READ-ONLY disk, then the no-write-permission field is set in the status word. Protocol errors will cause ERROR packets to be sent.

Writes can timeout. Again, clients are expected to deal with a WRITE timeout in the same way in which they would deal with a disk timeout.

4.2.9. Packet Format: WRITEACK

Format of WRITEACK Packet

<Byte 0>	<Byte 1>	<Byte 2>	<Byte 3>
Packet Type	Padding		RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Block Number			
Drive Status			
Number of Blocks for this ACK			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.9.1. Field Definitions for Packet Type: WRITEACK

- Block Number

  The number of the block that the server is acknowledging having written.

- Status

  The 32-bit Status word that represents the state of the disk and reports errors in the write procedure.

- Number of Blocks

  The number of blocks that have been successfully written.

4.2.9.2. Operation

WRITEACK is the response to the WRITE request. It signals the completion of the WRITE request. The WRITEACK is not sent until after the burst of write requests has been written to the physical disk. Only one WRITEACK is sent per burst.

Disk errors are reported through the status word.

4.2.10. Packet Format: ERROR

Format of ERROR Packet

< Byte 0 >	< Byte 1 >	< Byte 2 >	< Byte 3 >
Packet Type	Error Type	Padding	RVD Version
Drive			
Nonce			
Index			
Checksum			
Reserved			
Up to RVDDSIZE (512) Bytes of Error Dependent Data			

(See the section on RVD Packet Format for a complete description of the fields in the standard RVD header: Packet Type, RVD Version, Drive, Nonce, Index, and Checksum.)

4.2.10.1. Field Definitions for Packet Type: ERROR

- Error Type

  A byte representing the type of error that the ERROR packet is reporting.

4.2.10.2. Operation

ERROR packets are sent out when the server wants to tell the user that some error has occurred. They are usually sent when the error was in the protocol, or some other high level thing wrong with the virtual disk system. Errors that are roughly equivalent to those that a physical disk would give are typically returned in the 32 bit status word that is part of BLOCK and WRITEACK.

4.2.11. RVD Protocol Constants

```
RVDVERSION        4        /* Current protocol version */


/*
 * IP protocol number
 */
RVDPROTO  66

/* Packet types */
RVDSPIN           1        /* SPIN-UP packet */
RVDSDOWN  2                /* SPIN-DOWN packet */
RVDREAD           3        /* READ packet */
RVDWRITE  4                /* WRITE packet */
RVDSPACK  17               /* Ack for SPIN-UP */
RVDERROR  18               /* ERROR packet */
RVDACK            19       /* Ack for SPIN-DOWN */
RVDBLOCK  20               /* Block of data */
RVDWACK           21       /* Ack for WRITE */


/*
 * Status word masks
 */
RVDSTVAL   0001            /* If 0 then valid data */
RVDSTCNT   0036            /* Count of tries on foreign end - 1 */
RVDSTADR   0040            /* Bad block address */
RVDSTWRL   0100            /* Write attempted on read-only disk */


/*
 * Opening modes
 */
RVDMRO            0001     /* Read-only mode */
RVDMSHR           0002     /* Shared mode */
RVDMEXC           0004     /* Exclusive mode */


/*
 * Error types
 */
RVDENOER   0000            /* No error */
RVDEND            0001     /* Non-existent drive */
RVDEBPWD   0002            /* Bad password for mode */
RVDEOMD           0003     /* Already open in a different mode */
RVDECKSM   0004            /* Invalid Checksum */
RVDEIDX           0005     /* Index correction */
RVDEPACK   0006            /* Non-existent disk-pack */
RVDESPN           0007     /* Drive already spun up */
RVDEBMD           0010     /* Bad mode */
RVDEPKT           0011     /* Unknown packet type */
RVDENAH           0012     /* Non Active Host */
```

RVDEXMD	0013	/* Pack was spun up in EXCLUSIVE mode */
RVDEZBL	0014	/* Zero blocks requested */
RVDETBL	0015	/* Too many blocks requested */
RVDEPNM	0016	/* Pack not physically mounted */
RVDETCG	0017	/* Too many connections for this server */
RVDETGH	0020	/* Too many connections for this host */
RVDESNA	0021	/* Server not currently active */
RVDEIDA	0022	/* Identical pack already spun up in this drive, in the requested mode */
RVDERQU	0023	/* Requested mode unavailable. */
RVDETIM	0064	/* Timeout */
RVDEBVER	0065	/* Invalid version */

5. RVD CONTROL PROTOCOL SPECIFICATION

This section is optional reading.

5.1. Overview

The Remote Virtual Disk Control (RVDCTL) protocol supplies a Remote Virtual Disk server with both operating instructions and information about its configuration. An RVD server process comes into existence with no knowledge of the physical configuration of the system in which it is embedded or the logical configuration of the virtual disk packs it is to manage. These virtual disk packs may already exist. Supplying this information via a network connection instead of from files on the server host makes it possible to administer all aspects of server operation remotely.

This description assumes that the reader is already familiar with the basic concepts and terminology of the Remote Virtual Disk system, as described in Chapter 1 of this article.

* Operations and operands marked with an asterisk to the left have not yet been implemented.

There are four scenarios in which the RVDCTL protocol is used:

(1) Initialization

The first step after creating an RVD server process is to send it, using the RVDCTL protocol, a description of the physical and virtual disk configuration it is to manage. Because RVDCTL is a network protocol, the permanent data base that contains this state description may be managed on a machine different from that of the server.

(2) Permanent changes

When permanent changes to the physical and virtual disk pack configuration are desired, a management program both updates the permanent data base and sends to the server the same updates, again using the RVDCTL protocol.

(3) Temporary changes, for maintenance.

A client that can supply an operations password can invoke certain maintenance functions of the RVDCTL protocol, such as changing the logging level, shutting down the server, posting a message, or forcing off certain clients. For operations purposes, it is possible to invoke any of the update functions normally associated with permanent changes. Although a running RVD server would operate on the updated basis, if that server process were killed and recreated, such temporary changes would be forgotten, because the new server would receive its initialization from the permanent data base.

(4) Client use

RVD clients use the RVDCTL protocol for certain client-server interactions, such as flushing out old spinups, and inquiring about current operations.

There are several programs that invoke the RVDCTL protocol. Corresponding to the second scenario, above, is a data base management program (*vddb*(8)), which operates as follows:

- The person in charge of maintaining the data base runs the data base management program, which prompts for and validates input.
- The data base management program updates the disk files containing the permanent data base.
- The data base management program opens a control connection to the currently-running server and sends the server the updated information.

- The server modifies its state tables according to the requests.
- The server acknowledges the modification.

For simplicity, the data base management program stores its permanent data base in the form of a sequence of already-formatted protocol messages, so the initialization scenario is accomplished simply by sending a copy of the permanent data base to the server on the RVDCTL network connection. The *rvdsend*(8) program does this job.

The programs that perform the third and fourth scenarios are commands that can be run on any client (perhaps on behalf of another client) to invoke one or more specific RVDCTL functions at the server directly, without involving the database program.

Because most control operations for the RVD server require transferring only small amounts of data, and one wants to be able to implement servers on machines that do not provide a full TCP, the control protocol is UDP-based. It is a simple, lock-step, idempotent, message-response protocol. For all the control functions of interest, the control data fit into a single packet, which further simplifies the protocol. Idempotent means that if a client receives no response to a request (and is therefore unsure of whether or not the server acted on it), it is always safe to resend the same request, because by design successive repetitions of all RVDCTL operations have no ill effects.

The RVDCTL protocol carries very little traffic in comparison with the RVD protocol, so ease of construction and debugging therefore has a higher priority than performance. So that the control connection can handle operations of varying parameter requirements (to avoid the need to design a new packet format every time a new control function is added), and so that a single source implementation can apply to machines of different byte order, all data is transmitted as ASCII strings. For simplicity in parsing, arguments are transmitted in the format "keyword=value". This approach also makes RVDCTL packets self-explanatory when encountered during monitoring or auditing.

5.2. Syntax

```
<message>    :==   <opcode> <operands>
<opcode>     :==   operation= <value> |
                   success= <value> |
                   failure= <value> error= <value>
<operands>   :==   <operand> | <operands> <operand>
<operand>    :==   <keyword> = <value>
<keyword>    :==   <string>
<value>      :==   <string>
```

<*string*> is a string of network ASCII characters, terminated by a separator character. The separator characters are space, tab, newline, carriage return, and formfeed. One or more separators must appear between operands. Separators may be included in strings by quoting them. The quote character is the backslash (\). A backslash may be included in an string by doubling it (\). Also, to include an equals sign (=) in a string, it must be quoted.

The particular keywords used depends on the operation being invoked. The keywords "operation", "password", "nonce", "success", "failure", and "error" are universal.

(1) The "operation" keyword must be the first keyword in each request packet. Its value is the name of the requested operation.

(2) The "password"keyword operation requires one. There are three kinds of passwords. The operations password authorizes shutdown, logging, and physical configuration management. The administrative password authorizes allocation and deallocation of virtual disks. Individual pack passwords authorize usage of

those packs.

*

(3) The "nonce" keyword appears in every request, with a value chosen by the requester to be different from any other request for which a late response might still arrive. Every response contains a copy of the nonce of the request to which it responds.

(4) The "success" keyword must be the first keyword in a response to a successful request. Its value is the name of the operation performed.

(5) The "failure" keyword must be the first keyword in a response to an unsuccessful request. Its value is the name of the operation that failed. It may be accompanied by an "error=" operand describing the error which occurred. The value of the "error" keyword is a human-readable string describing the error which occurred.

Except for "operation", "success", "failure", be the first keyword in a message, the order of operands in a message is unimportant.

Where a number is called for, it is represented in the operand value string as an ASCII decimal integer. Where an Internet Protocol (IP) Address is called for, it is represented in the operand value string as .q "A.B.C.D", in network standard ASCII decimal form. Where a mode is called for, it is represented in the operand value string as an ASCII decimal number coded in the following way, in any sum desired:

1 =	read-only spinups allowed
2 =	shared spinups allowed (not currently implemented)
4 =	exclusive spinups allowed
0 =	no spinups allowed

Port: The RVDCTL protocol operates on UDP port 531.

5.3. Operations

(1) Add a physical device partition to the set of partitions managed by the RVD server.

operation=add_physical

Required operands:

password	The operations password for the RVD server.
filename	Path name of the device to be managed as a physical partition.
blocks	The number of 512-byte sectors in this physical partition.

The device need not be a real physical disk; any device (e.g., a file) that behaves like a raw disk partition will work equally well.

* If the server finds it is unable to open the physical device it marks the physical device as "disused", and returns an error. (See *disuse_physical.*)

Note that *add_physical* is normally invoked as part of updating the permanent data base that describes the server configuration. If *add_physical* is invoked without a data base update, the next time the server is shut down the change made by the add_physical operation will be forgotten.

*

(2) Delete a physical device partition from the set of partitions managed by the RVD server.

operation = delete_physical

Required operands:

password	The operations password for the RVD server.
filename	Path name of the device to be managed as a physical partition.

If there are any virtual disk packs allocated on this physical device, delete_physical returns an error response, and does not delete the device.

Note that the delete_physical operation is normally invoked as part of updating the permanent data base that describes the server configuration. If delete_physical is invoked without a data base update, the next time the server is shut down the change made by the delete_physical operation will be forgotten.

*

(3) Stop using a physical disk partition.

operation = disuse_physical

Required operands:

password	The operations password for this RVD server.
physical	The pathname of the device partition to be disused.

The *disuse_physical* operation allows an operator to take a partition out of use, for example because the disk is getting hard errors. (The server may, on its own, place a partition that is getting errors in disused mode.) *Add_virtual* and *delete_virtual* operations may be executed on a disused partition. Attempts to spinup packs that are located on a disused partition receive the error response "pack temporarily unavailable." The server continues to maintain records of existing connections and to allow spindowns, but attempts to read or write a previously spunup pack receive an error packet containing the error code "pack temporarily unavailable."

*

(4) Try to use a physical disk partition.

operation = use_physical

Required operands:

password	The operations password for this RVD server.
physical	The pathname of the device partition to be used.

If a physical partition is currently disused, this operation puts the partition back into service. If the physical partition does not exist or is already in use, *use_physical* returns an error.

(5) Allocate a virtual disk pack.

operation = add_virtual

Required operands:

password	The administrative password for this RVD server.
physical	The pathname of the device partition this virtual disk pack is to be on.
name	The name of this virtual disk pack (n.b., upper and lower case are distinguished.)
packid *	The unique id of this pack on this server.
owner	The name of this virtual disk pack"s owner.
rocap	The read-only mode password (may be null).
excap	The exclusive mode password (may be null).
shcap	The shared mode password (may be null).
modes	The allowable modes this virtual disk pack may be spun up in.
offset	The offset, in blocks, of this virtual disk pack from the start of the physical partition.
blocks	The number of 512-byte blocks in this virtual disk.

Optional operands:

ownhost	Internet address of the owning host of this virtual disk pack. If none is supplied, the disk is assumed to not have an owning host.

Add_virtual is normally invoked as part of updating the permanent data base that describes the server configuration. If *add_virtual* is invoked without a data base update, the next time the server is shut down the addition made by *add_virtual* operation will be forgotten.

(6) Deallocate a virtual disk pack

operation = delete_virtual

Required operands:

password	Administrative password.

Optional operands:

packid *	The unique identifier of the virtual disk pack to be deallocated
name	The name of the virtual disk pack to be deallocated.

One of the operands {packid, name} must be present. If both are present, they must refer to the same pack.

Delete_virtual is normally invoked as part of updating the permanent data base that describes the server configuration. If *delete_virtual* is invoked without a data base update, the next time the server is shut down the deletion made by *delete_virtual* operation will be forgotten.

(7) Modify the definition of a virtual disk pack.

operation = modify_virtual

Required operands:

password	Administrative password.
name	The name of the virtual disk pack whose description is to be modified.

Optional operands (any operand present supersedes the value previously supplied by *add_virtual* or *modify_virtual* of the corresponding parameter for this virtual disk pack):

packid *	The unique identifier of this pack. If provided, this operand is used to identify pack to be modified, and the name operand is taken to be a new pack name.
owner	The name of this virtual disk pack"s owner.
rocap	The read-only mode password
excap	The exclusive mode password
shcap	The shared mode password
modes	The allowable modes this virtual disk pack may be spun up in, as an ASCII decimal number.
blocks	The number of 512-byte blocks in this virtual disk. Must be less than or equal to the current number of blocks on this disk. In general, changing a disk"s size is a bad idea, especially if it is currently in use.
ownhost	Internet address of the owning host of this disk.

Modify_virtual is normally invoked as part of updating the permanent data base that describes the server configuration. If *modify_virtual* is invoked without a data base update, the next time the server is shut down the changes made by the *modify_virtual* operation will be forgotten.

(8) Exchange the names of two virtual disk packs.

operation = exchange_names

Required operands:

name1	Desired name for the first virtual disk pack
packid1 *	Unique identifier of first pack
password1	The exclusive mode password of the first virtual disk pack
name2	Desired name for the second virtual disk pack
packid2 *	Unique identifier of second pack
password2	The exclusive mode password of the second

The operands *name1* and *name2* must be the names presently associated with the two packs. Success for this operation means that those two names are now associated with the packs in the order requested, whether or not they were before the operation.

This operation is used as part of an update procedure, in which two copies of a library virtual disk pack are maintained. One copy is normally spun up by clients in read-only mode; the other is the "maintenance" copy, to which the owner makes changes. Once a consistent set of changes are ready for release, the owner exchanges the names of the packs. Other users can then spin the pack down and back up again by name to get the new copy. If the server shuts down and restarts, clients that have temporarily cached the packid can respin up the old pack by packid, to complete their session without being forced prematurely to switch to the new library.

Exchange_names is normally invoked as part of updating the permanent data base that describes the server configuration. If *exchange_names* is invoked without a data base update, the next time the server is shut down *exchange_names* will be forgotten.

(9) Force a virtual disk pack to be spun down.

operation = spindown_virtual

Required operands:

name	The name of the virtual disk pack to be forced down.
password	The exclusive mode password of the virtual disk pack to be forced down.

This operation is normally used by the owner of a virtual disk pack that was spun up on a machine that crashed. It forces the specified virtual disk pack to be spun down from all the machines that have it spun up.

(10) Force all virtual disk packs of a given client at this server to be spun down.

operation = spindown_host

Required operands:

name	The Internet address of the client whose disk packs are to be forced down.

Optional operands:

password	If the spindown_host request was not sent from the client whose disks are to be spun down, the operations password must be supplied.

This operation has two uses:

a) It should appear in a 4.2/RT client's /etc/rc file, or a DOS client"s autoexec.bat file, so that when a host recovers from a crash, all its previously spunup virtual disk packs are spun down. This spindown insures that the server state agrees with the client state.

b) An operator can use this operation to force down the virtual disks of a client which has crashed and that may be down for some time.

(11) Display all spinups involving a virtual disk pack, or a client.

operation = display_virtual

Required operands (exactly one of the following must be present):

name	The name of a virtual disk pack. If present, *display_virtual* returns a list of all the spinups of this disk pack. These are the spinups that would be forced down if a *spindown_virtual* operation naming this pack were performed.
host	The IP address of a client. If present, *display_virtual* returns a list of all the spinups of this client. These are the spinups that would be forced down if a *spindown_host* operation naming this client were performed.

Optional operand:

start = < value >	An ASCII decimal integer giving the offset of the first spinup description wanted. This operand is normally supplied if a previous invocation of display_virtual contained the response "more = true".
password	If the display_virtual operation requests information about a client different from the one making the request, the operations password must be supplied.

This operation returns a success packet containing an ASCII text string describing the spinups (host/drive number pairs) of this virtual disk. The response packet contains:

```
success = display_virtual
number = < value1 >
connections = < value2 >
more = true                    (optional response)
```

< value1 >
The number of currently active spinups for this virtual disk pack or client.

< value2 >
A canonicalized string, with one line per spinup, containing as many spinup descriptions as will fit in one UDP packet. Each line is a collection of space-separated tokens, as follows:

pack = library host = 18.72.0.5 drive = 9 mode = 4

Since the string is canonicalized, all spaces and CRLF sequences are quoted.

If there were more spinup descriptions than would fit in a single packet, the response operand "more = true" will appear.

(1) Log statistics of external interactions.

operation = log_external_statistics

Required operand:

password	The operations password

Dump into the log file all statistics kept by the RVD server concerning interactions with clients--number of packets exchanged, disk operations, etc.)

(2) Log all statistics

operation = log_all_statistics

Required operand:

password	The operations password

Dump into the log file all statistics kept by the rvd server.

(3) Shut down server

operation = shutdown

Required operands:

password	The operations password

Log all statistics, then perform a clean shutdown of the server.

(4) Change log level

operation = log_level

Required operands:

password	The operations password
level	New log level as a hex number (N.B., not decimal.)

Change which events are logged; see specification of the RVD protocol for definition of log levels.

(5) Truncate log

operation = log_truncate

Required operands:

password	The operations password

Truncate the log file to keep it from growing too large. [In the BSD 4.3 UNIX implementation of RVD, logging is done with the UNIX logging system (syslogd), so this operation has no effect.]

(6) Allow spinups

operation = allow_spinups

Required operands:

password	The operations password, for a physical device, or the exclusive mode pack password, for a single virtual pack.
mode	The mode of allowed spinups.

Optional operands:

physical	Path name of the device partition to which this mode setting applies. (If absent, the mode applies to all partitions managed by this server.)

name	The name of a virtual disk pack to which this mode setting applies.

Response operand:

oldmode	The spinup mode that was formerly allowed for this partition or virtual pack.

This operation is used to prevent or allow further spinups of a single virtual pack, or all the virtual packs on a given device partition of this RVD server; it has no effect on spinups already in force. When a server first comes up it allows no spinups (mode = 0), so an invocation of *allow_spinups* is required as part of starting a server.

A separate allowed spinup mode value is maintained for each pack and for each partition; the actual modes permitted for a pack are given by the logical AND of the mode value for the pack and the mode value for the partition on which it is located.

The server rejects spinups that would be allowed by the static pack description but that are prevented by the current setting of *allow_spinups* with a distinct error code indicating temporary unavailability.

Usage scenarios: If a server is to be dumped, one might allow only read spinups during the dump; if a server is to be taken down one might sometime earlier allow no new spinups. The maintainer of a library disk pack that needs to be updated might first allow no spinups, then after a period of time adequate for most clients to finish their sessions, do a *spindown_virtual* to get rid of any remaining spinups.

(7) Post an operations message.

operation = set_message

Required operands

password	The operations password
message = < string >	The (canonicalized) message < string > replaces any previous operations message. If < string > is null, any previous message is cleared. The content of the message is limited to 400 bytes, and is network ASCII (lines terminated with canonicalized CRLF"s).

This operation, together with the next one, allows an operator to post a message (e.g., "server going down at 5:00 p.m. for preventive maintenance") for clients of an RVD server.

(8) Get the operations message.

operation = get_message (no required or optional operands)

Response operands:

success = get_message	
message = < string >	< string > is a canonicalized string of network ascii to be displayed as an operations message. If there is no current operations message, < string > is null. (Note that in either case < string > is terminated by an operand separator.)

This operation would normally be invoked by a client as part of bringing up a system that uses RVD and also whenever spinning up a virtual disk pack.

*

(9) Change a user password.

operation = change_password

Required operands

packname	The name of the virtual disk pack whose password is to be changed.
mode	The spinup modes for which a new password is being supplied. If more than one mode is specified, the operation will be rejected unless the old passwords for the several modes are all the same as the old_password operand.
old_password	The current password for this pack and mode; a null string if there is no current password.
new_password	The new password; a null string if there is to be no password.

Note that this function is not intended for direct use by a client, but rather for use by the database update system; if used by a client without also updating the database, the password will be restored to its old value the next time the RVD server is restarted.

(10) Return a list of active virtual packs

operation = display_active

Optional operands:

filename	Path name of device partition for which a list of active virtual packs is wanted. If omitted, a list of all active virtual packs is returned.
start = < value >	A number giving the offset of the first information line wanted. This operand is normally supplied if the previous invocation of display_active included the response operand "more = true".

Response operand:

number = < value1 >
activity = < value2 >
more = true (optional response)

< value1 >
The number of currently active packs on this partition or, if no partition was specified, on this server.

< value2 >
A single canonicalized netascii string containing one line of information for each

active virtual pack. A typical line looks like:

partition = /dev/ra0g pack = library mode = 1 connections = 5 idle = 1721

If there were more activity descriptions than would fit in a single packet, the response operand "more = true" will appear.

Idle time is measured in seconds since most recent access. Note that the idle time is purely an activity hint, to determine whether or not a pack that appears to be spun up is actively in use. It is maintained by the server only to a rough approximation.

*

(11) Obtain server load statistics

operation = get_load

Required operands:

password	The operations password for the RVD server.

Response:

load = < string >	< string > is a canonicalized netascii string containing load statistics ready for display.

(12) Change authorization for operations and administrative operations.

operation = require_authorization		(no required or optional operands)

When an RVD server begins operation, it accepts RVD control protocol requests only from the same host on which it is operating, and it does not require operations or administrative passwords. (Starting without passwords allows automating initialization without the need to store those passwords in clear form.) The *require_authorization* operation causes the server to read operations and administrative passwords from a file in the file system of the server's host. After *require_authorization* is executed all operations listed above as requiring either an administrative or operations password do actually require them. Whenever *require_authorization* is invoked, the RVD server reinitializes its copy of the operations and administrative passwords from */etc/rvdauthor*.

There are two scenarios of use of require_authorization. The first is at system initialization time:

- start server
- send initializing control sequences, if any
- send require_authorization
- await success of require_authorization
- declare initialization successful.

The second scenario is to change the operations or administrative passwords.

- modify file containing operations and maintenance passwords.
- send require_authorization

APPENDICES

The following appendices are provided:

- **Appendix A. Software Description**

  contains a list of functions supported in this distribution.

- **Appendix B. Graphics Manual Pages**

  contains manual pages for the graphics routines used by the C subroutine interface described in Volume II.

- **Appendix C. High C Programmer's Guide**

  contains a guide for programming in C, using the High C compiler from MetaWare Incorporated.

This page intentionally left blank.

Appendix A. Software Description

This appendix contains listings of supported and unsupported functions found in 4.2/RT.

1. SUPPORTED FUNCTIONS

The following sections list the functions supported in this distribution. Items marked with an asterisk (*) are 4.2/RT functions not found in 4.2BSD. Items marked with a dagger (†) were developed or revised at the University of California at Berkeley after the release of 4.2BSD.

1.1. Section 1: Commands and Application Programs.

	Man Page	Name	Section and Description
	adb	adb	(1) debugger
	addbib	addbib	(1) create or extend bibliographic database
*	aedjournal	aedjournal	(1) display commands in a log file
*	aedrunner	aedrunner	(1) execute graphics commands in a log file
	apply	apply	(1) apply a command to a set of arguments
	apropos	apropos	(1) locate commands by keyword lookup
	ar	ar	(1) archive and library maintainer
	as	as	(1) assembler
†	at	at	(1) execute commands at a later time
†	atq	atq	(1) print the queue of jobs waiting to be run
†	atrm	atrm	(1) remove the jobs spooled by at
	awk	awk	(1) pattern scanning and processing language
	basename	basename	(1) strip filename affixes
	bc	bc	(1) arbitrary-precision arithmetic language
	biff	biff	(1) be notified if mail arrives and sender
	binmail	binmail	(1) send or receive mail among users
*	bitprt	bitprt	(1) capture the image on a bitmap display and print it on an IBM printer
	cal	cal	(1) print calendar
	calendar	calendar	(1) reminder service
†	cat	cat	(1) concatenate and print
	cb	cb	(1) C program beautifier
	cc	cc	(1) default C compiler
	cd	cd	(1) change working directory
	checknr	checknr	(1) check nroff/troff files
	chfn	chfn	(1) change finger entry
†	chgrp	chgrp	(1) change group
†	chmod	chmod	(1) change mode
	chsh	chsh	(1) change default login shell
	clear	clear	(1) clear terminal screen
	cmp	cmp	(1) compare two files
	col	col	(1) filter reverse line feeds
	colcrt	colcrt	(1) filter nroff output for CRT previewing
*	colpro	colpro	(1) column filter for IBM 4201 Proprinter
	colrm	colrm	(1) remove columns from a file
	comm	comm	(1) select or reject lines common to two sorted files
	compact	ccat	(1) compress and uncompress files and cat them
	compact	compact	(1) compress and uncompress files and cat them
	compact	uncompact	(1) compress and uncompress files and cat them

†	compress	uncompress	(1) compress and expand data
†	compress	zcat	(1) compress and expand data
†	cp	cp	(1) copy
	crypt	crypt	(1) encode/decode
	csh	csh	(1) a shell (command interpreter) with C-like syntax
†	ctags	ctags	(1) create a tagsfile
	date	date	(1) print and set the date
	dbx	dbx	(1) debugger
	dc	dc	(1) desk calculator
†	dd	dd	(1) convert and copy a file
	deroff	deroff	(1) remove nroff,troff, tbl and eqn constructs
	df	df	(1) disk free
	diction	diction	(1) print wordy sentences; thesaurus for diction
	diction	explain	(1) print wordy sentences; thesaurus for diction
	diff	diff	(1) differential file and directory comparator
	diff3	diff3	(1) 3-way differential file comparison
*	dosread	dosread	(1) read, write, dir, delete on PC-DOS diskette
	du	du	(1) summarize disk usage
*	dumpaed	dumpaed	(1) dump aed display memory as a binary file
*	dumpapa16	dumpapa16	(1) dump apa16 display memory as a binary file
*	dumpapa8	dumpapa8	(1) dump apa8 display memory as a binary file
*	dumpapa8c	dumpapa8c	(1) dump apa8c display memory as a binary file
	echo	echo	(1) echo arguments
	ed	ed	(1) text editor
	efl	efl	(1) Extended FORTRAN Language
	eqn	checkeq	(1) typeset mathematics
	eqn	eqn	(1) typeset mathematics
	eqn	neqn	(1) typeset mathematics
†	error	error	(1) analyze and disperse compiler error messages
	ex	edit	(1) text editor
	ex	ex	(1) text editor
	expand	expand	(1) expand tabs to spaces and vice versa
	expand	unexpand	(1) expand tabs to spaces and vice versa
	expr	expr	(1) evaluate arguments as expressions
	eyacc	eyacc	(1) modified yacc allowing much improved error recovery
	f77	f77	(1) FORTRAN 77 compiler
	false	false	(1) provide truth values
	false	true	(1) provide truth values
	file	file	(1) determine file type
	find	find	(1) find files
	finger	finger	(1) user information lookup program
	fmt	fmt	(1) simple text formatter
	fold	fold	(1) fold long lines for finite width output device
	fpr	fpr	(1) print FORTRAN file
†	from	from	(1) from whom is my mail?
	fsplit	fsplit	(1) split a multi-routine FORTRAN file into individual files
	ftp	ftp	(1C) file transfer program
	gcore	gcore	(1) get core images of running processes
	gprof	gprof	(1) display call graph profile data
	graph	graph	(1G) draw a graph
	grep	egrep	(1) search a file for a pattern
	grep	fgrep	(1) search a file for a pattern
	grep	grep	(1) search a file for a pattern

	groups	groups	(1)	show group memberships
*	hc	hc	(1)	High C compiler
	head	head	(1)	give first few lines
†	hostid	hostid	(1)	set or print identifier of current host system
	hostname	hostname	(1)	set or print name of current host system
	indent	indent	(1)	indent and format C program source
	install	install	(1)	install binaries
	intro	intro	(1)	introduction to commands
	iostat	iostat	(1)	report I/O statistics
	join	join	(1)	relational database operator
*	kbdlock	kbdlock	(1)	lock the keyboard of the IBM RT PC
	kill	kill	(1)	terminate a process with extreme prejudice
	last	last	(1)	indicate last logins of users and teletypes
	lastcomm	lastcomm	(1)	show last commands executed in reverse order
	ld	ld	(1)	link editor
†	learn	learn	(1)	computer-aided instruction about UNIX
†	leave	leave	(1)	remind you when you have to leave
	lex	lex	(1)	generator of lexical analysis programs
	lint	lint	(1)	a C program verifier
†	ln	ln	(1)	make links
†	lock	lock	(1)	reserve a terminal
	login	login	(1)	sign on
	look	look	(1)	find lines in a sorted list
	lookbib	indxbib	(1)	build inverted index for a bibliography
	lookbib	lookbib	(1)	find references in a bibliography
	lorder	lorder	(1)	find ordering relation for an object library
	lpq	lpq	(1)	spool queue examination program
	lpr	lpr	(1)	off-line print
	lprm	lprm	(1)	remove jobs from the line printer spooling queue
	ls	ls	(1)	list contents of directory
	m4	m4	(1)	macro processor
†	mail	mail	(1)	send and receive mail
†	make	make	(1)	maintain program groups
†	man	man	(1)	find manual information by keywords; print out the manual
	mesg	mesg	(1)	permit or deny messages
	mkdir	mkdir	(1)	make a directory
	mkstr	mkstr	(1)	create an error message file by massaging C source
†	more	more	(1)	file perusal filter for CRT viewing
†	more	page	(1)	file perusal filter for CRT viewing
	msgs	msgs	(1)	system messages and junk mail program
†	mset	mset	(1)	ASCII to IBM 3270 keyboard map
	mt	mt	(1)	magnetic tape manipulating program
	mv	mv	(1)	move or rename files
	netstat	netstat	(1)	show network status
	newaliases	newaliases	(1)	rebuild the data base for the mail aliases file
	nice	nice	(1)	run a command at low priority (sh only)
	nice	nohup	(1)	run a command at low priority (sh only)
	nm	nm	(1)	print name list
	nroff	nroff	(1)	text formatting
	od	od	(1)	octal, decimal, hex, ASCII dump
	pagesize	pagesize	(1)	print system page size
	passwd	passwd	(1)	change login password

*	pcc	pcc	(1) pcc-based C compiler
*	pf	pf	(1) set keyboard program-function keys
*	pic	pic	(1) troff preprocessor for drawing simple pictures (part of optional ditroff feature)
	plot	plot	(1G) graphics filters
*	pp	pp	(1) Professional Pascal compiler (part of optional Professional Pascal feature)
	pr	pr	(1) print file
*	prfl	prfl	(1) IBM 4201 Proprinter/IBM 5152 Graphics Printer nroff post-processing filter
	print	print	(1) pr to the line printer
	printenv	printenv	(1) print out the environment
	prmail	prmail	(1) print out mail in the post office
	prof	prof	(1) display profile data
*	proff	proff	(1) nroff for the IBM 5152 Graphics Printer and IBM 4201 Proprinter
	ps	ps	(1) process status
	pti	pti	(1) phototypesetter interpreter
*	ptroff	ptroff	(1) print troff files on IBM 3812 Pageprinter (part of optional ditroff feature)
	ptx	ptx	(1) permuted index
	pwd	pwd	(1) working directory name
	quota	quota	(1) display disc usage and limits
	ranlib	ranlib	(1) convert archives to random libraries
	ratfor	ratfor	(1) Rational FORTRAN dialect
	rcp	rcp	(1C) remote file copy
	refer	refer	(1) find and insert literature references in documents
	reset	reset	(1) reset the teletype bits to a sensible state
	rev	rev	(1) reverse lines of a file
	rlogin	rlogin	(1C) remote login
	rm	rm	(1) remove (unlink) files or directories
	rm	rmdir	(1) remove (unlink) files or directories
	rmail	rmail	(1) handle remote mail received via uucp
	rmdir	rm	(1) remove (unlink) directories or files
	rmdir	rmdir	(1) remove (unlink) directories or files
	roffbib	roffbib	(1) run off bibliographic database
	rsh	rsh	(1C) remote shell
	ruptime	ruptime	(1C) show host status of local machines
	rwho	rwho	(1C) who is logged in on local machines
*	scale	scale	(1) resize a bit image
	script	script	(1) make typescript of terminal session
	sed	sed	(1) stream editor
	sendbug	sendbug	(1) mail a system bug report to 4bsd-bugs
	sh	sh	(1) command language
	size	size	(1) size of an object file
	sleep	sleep	(1) suspend execution for an interval
	soelim	soelim	(1) eliminate .so's from nroff input
	sort	sort	(1) sort or merge files
	sortbib	sortbib	(1) sort bibliographic database
	spell	spell	(1) find spelling errors
	spell	spellin	(1) find spelling errors
	spell	spellout	(1) find spelling errors
	spline	spline	(1G) interpolate smooth curve

	split	split	(1) split a file into pieces
	strings	strings	(1) find the printable strings in an object or other binary file
	strip	strip	(1) remove symbols and relocation bits
	struct	struct	(1) structure FORTRAN programs
	stty	stty	(1) set terminal options
	style	style	(1) analyze surface characteristics of a document
	su	su	(1) substitute user id temporarily
	sum	sum	(1) sum and count blocks in a file
	symorder	symorder	(1) rearrange name list
	sysline	sysline	(1) display system status on status line of a terminal
	tabs	tabs	(1) set terminal tabs
†	tail	tail	(1) deliver the last part of a file
	talk	talk	(1) talk to another user
	tar	tar	(1) tape archiver
	tbl	tbl	(1) format tables for nroff or troff
	tee	tee	(1) pipe fitting
†	telnet	telnet	(1C) user interface to the TELNET protocol
	test	test	(1) condition command
†	tftp	tftp	(1C) trivial file transfer program
	time	time	(1) time a command
†	tip	cu	(1C) connect to a remote system
†	tip	tip	(1C) connect to a remote system
†	tn3270	tn3270	(1) full-screen remote login to IBM VM/CMS
	touch	touch	(1) update file date last modified
	tr	tr	(1) translate characters
	trman	trman	(1) translate version 6 manual macros to version 7 macros
	troff	nroff	(1) text formatting and typesetting
	troff	troff	(1) text formatting and typesetting
	true	false	(1) provide truth values
	true	true	(1) provide truth values
	tset	tset	(1) terminal dependent initialization
	tsort	tsort	(1) topological sort
	tty	tty	(1) get terminal name
	ul	ul	(1) do underlining
†	unifdef	unifdef	(1) remove ifdef'ed lines
	uniq	uniq	(1) report repeated lines in a file
	units	units	(1) conversion program
*	up	down	(1) client Remote Virtual Disk (RVD) utilities
*	up	up	(1) client Remote Virtual Disk (RVD) utilities
	uptime	uptime	(1) show how long system has been up
	users	users	(1) compact list of users who are on the system
	uucp	uucp	(1C) UNIX to UNIX copy
	uucp	uulog	(1C) UNIX to UNIX copy
	uuencode	uudecode	(1C) encode/decode a binary file for transmission via mail
	uuencode	uuencode	(1C) encode/decode a binary file for transmission via mail
	uusend	uusend	(1C) send a file to a remote host
	uux	uux	(1C) UNIX to UNIX command execution
†	vacation	vacation	(1) return "I am on vacation" indication
†	vgrind	vgrind	(1) grind nice listings of programs for the IBM 3812 Pageprinter
	vi	vi	(1) screen oriented (visual) display editor based on ex
	vmstat	vmstat	(1) report virtual memory statistics

	w	w	(1) who is on and what they are doing
	wait	wait	(1) await completion of process
	wall	wall	(1) write to all users
	wc	wc	(1) word count
	what	what	(1) show what versions of object modules were used to construct a file
	whatis	whatis	(1) describe what a command is
	whereis	whereis	(1) locate source, binary, and/or manual for program
	which	which	(1) locate a program file including aliases and paths (csh only)
	who	who	(1) who is on the system
	whoami	whoami	(1) print effective current user id
†	write	write	(1) write to another user
	xsend	enroll	(1) secret mail
	xsend	xget	(1) secret mail
	xsend	xsend	(1) secret mail
	xstr	xstr	(1) extract strings from C programs to implement shared strings
	yacc	yacc	(1) yet another compiler-compiler
	yes	yes	(1) be repetitively affirmative

1.2. Section 2: System Calls

	Man Page	Name	Section and Description
	accept	accept	(2) accept a connection on a socket
	access	access	(2) determine accessibility of file
	acct	acct	(2) turn accounting on or off
	bind	bind	(2) bind a name to a socket
	brk	brk	(2) change data segment size
	brk	sbrk	(2) change data segment size
	chdir	chdir	(2) change current working directory
	chmod	chmod	(2) change mode of file
	chmod	fchmod	(2) change mode of file
	chown	chown	(2) change owner and group of a file
	chown	fchown	(2) change owner and group of a file
	chroot	chroot	(2) change root directory
	close	close	(2) delete a descriptor
	connect	connect	(2) initiate a connection on a socket
	creat	creat	(2) create a new file
	dup	dup	(2) duplicate a descriptor
	dup	dup2	(2) duplicate a descriptor
	execve	execve	(2) execute a file
	exit	_exit	(2) terminate a process
	fcntl	fcntl	(2) file control
	flock	flock	(2) apply or remove an advisory lock on an open file
	fork	fork	(2) create a new process
	fsync	fsync	(2) synchronize a file's in-core state with that on disk
	getdtablesize	getdtablesize	(2) get descriptor table size
*	getfpemulator	getfpemulator	(2) return address of the floating point emulator if no FPA hardware is present
	getgid	getegid	(2) get group identity
	getgid	getgid	(2) get group identity

	getgroups	getgroups	(2)	get group access list
	gethostid	gethostid	(2)	get/set unique identifier of current host
	gethostid	sethostid	(2)	get/set unique identifier of current host
	gethostname	gethostname	(2)	get/set name of current host
	gethostname	sethostname	(2)	get/set name of current host
	getitimer	getitimer	(2)	get/set value of interval timer
	getitimer	setitimer	(2)	get/set value of interval timer
	getpagesize	getpagesize	(2)	get system page size
	getpeername	getpeername	(2)	get name of connected peer
	getpgrp	getpgrp	(2)	get process group
	getpid	getpid	(2)	get process identification
	getpid	getppid	(2)	get process identification
	getpriority	getpriority	(2)	get/set program scheduling priority
	getpriority	setpriority	(2)	get/set program scheduling priority
	getrlimit	getrlimit	(2)	control maximum system resource consumption
	getrlimit	setrlimit	(2)	control maximum system resource consumption
	getrusage	getrusage	(2)	get information about resource utilization
	getsockname	getsockname	(2)	get socket name
	getsockopt	getsockopt	(2)	get and set options on sockets
	getsockopt	setsockopt	(2)	get and set options on sockets
	gettimeofday	gettimeofday	(2)	get/set date and time
	gettimeofday	settimeofday	(2)	get/set date and time
	getuid	geteuid	(2)	get user identity
	getuid	getuid	(2)	get user identity
	intro	intro	(2)	introduction to system calls and error numbers
	ioctl	ioctl	(2)	control device
	kill	kill	(2)	send signal to a process
	killpg	killpg	(2)	send signal to a process group
	link	link	(2)	make a hard link to a file
	listen	listen	(2)	listen for connections on a socket
	lseek	lseek	(2)	move read/write pointer
	mkdir	mkdir	(2)	make a directory file
	mknod	mknod	(2)	make a special file
	mount	mount	(2)	mount or remove file system
	mount	umount	(2)	mount or remove file system
	open	open	(2)	open a file for reading or writing, or create a new file
	pipe	pipe	(2)	create an interprocess communication channel
	profil	profil	(2)	execution time profile
†	ptrace	ptrace	(2)	process trace
	quota	quota	(2)	manipulate disk quotas
	read	read	(2)	read input
	read	readv	(2)	read input
	readlink	readlink	(2)	read value of a symbolic link
	reboot	reboot	(2)	reboot system or halt processor
	recv	recv	(2)	receive a message from a socket
	recv	recvfrom	(2)	receive a message from a socket
	recv	recvmsg	(2)	receive a message from a socket
	rename	rename	(2)	change the name of a file
	rmdir	rmdir	(2)	remove a directory file
	select	select	(2)	synchronous I/O multiplexing
	send	send	(2)	send a message from a socket
	send	sendmsg	(2)	send a message from a socket
	send	sendto	(2)	send a message from a socket

	setgroups	setgroups	(2) set group access list
	setpgrp	setpgrp	(2) set process group
	setquota	setquota	(2) enable/disable quotas on a file system
	setregid	setregid	(2) set real and effective group ID
	setreuid	setreuid	(2) set real and effective user ID
	shutdown	shutdown	(2) shut down part of a full-duplex connection
	sigblock	sigblock	(2) block signals
	sigpause	sigpause	(2) automatically release blocked signals and wait for interrupt
	sigsetmask	sigsetmask	(2) set current signal mask
	sigstack	sigstack	(2) set and/or get signal stack context
	sigvec	sigvec	(2) software signal facilities
	socket	socket	(2) create an endpoint for communication
	socketpair	socketpair	(2) create a pair of connected sockets
	stat	fstat	(2) get file status
	stat	lstat	(2) get file status
	stat	stat	(2) get file status
	swapon	swapon	(2) add a swap device for interleaved paging/swapping
	symlink	symlink	(2) make symbolic link to a file
	sync	sync	(2) update super-block
	syscall	syscall	(2) indirect system call
	truncate	ftruncate	(2) truncate a file to a specified length
	truncate	truncate	(2) truncate a file to a specified length
	umask	umask	(2) set file creation mode mask
	unlink	unlink	(2) remove directory entry
	utimes	utimes	(2) set file times
*	vdspin	vdspin	(2) spin up or spin down a Remote Virtual Disk (RVD)
*	vdspin	vdspind	(2) spin up or spin down a Remote Virtual Disk (RVD)
*	vdstats	vdstats	(2) acquire client Remote Virtual Disk (RVD) statistics
	vfork	vfork	(2) spawn new process in a virtual memory efficient way
	vhangup	vhangup	(2) virtually "hangup" the current control terminal
	wait	wait	(2) wait for process to terminate
	wait	wait3	(2) wait for process to terminate
	write	write	(2) write on a file
	write	writev	(2) write on a file

1.3. Section 3: C Library Subroutines

Man Page	Name	Section and Description
abort	abort	(3) generate a fault
abs	abs	(3) integer absolute value
atof	atof	(3) convert ASCII to numbers
atof	atoi	(3) convert ASCII to numbers
atof	atol	(3) convert ASCII to numbers
bstring	bcmp	(3) bit and byte string operations
bstring	bcopy	(3) bit and byte string operations
bstring	bzero	(3) bit and byte string operations
bstring	ffs	(3) bit and byte string operations
crypt	crypt	(3) DES encryption
crypt	encrypt	(3) DES encryption
crypt	setkey	(3) DES encryption
ctime	asctime	(3) convert date and time to ASCII

	ctime	ctime	(3)	convert date and time to ASCII
	ctime	gmtime	(3)	convert date and time to ASCII
	ctime	localtime	(3)	convert date and time to ASCII
	ctime	timezone	(3)	convert date and time to ASCII
†	ctype	isalnum	(3)	character classification macros
†	ctype	isalpha	(3)	character classification macros
†	ctype	isascii	(3)	character classification macros
†	ctype	iscntrl	(3)	character classification macros
†	ctype	isdigit	(3)	character classification macros
†	ctype	isgraph	(3)	character classification macros
†	ctype	islower	(3)	character classification macros
†	ctype	isprint	(3)	character classification macros
†	ctype	ispunct	(3)	character classification macros
†	ctype	isspace	(3)	character classification macros
†	ctype	isupper	(3)	character classification macros
†	ctype	isxdigit	(3)	character classification macros
†	ctype	toascii	(3)	character classification macros
†	ctype	tolower	(3)	character classification macros
†	ctype	toupperisupper	(3)	character classification macros
	directory	closedir	(3)	directory operations
	directory	opendir	(3)	directory operations
	directory	readdir	(3)	directory operations
	directory	rewinddir	(3)	directory operations
	directory	seekdir	(3)	directory operations
	directory	telldir	(3)	directory operations
	ecvt	ecvt	(3)	output conversion
	ecvt	fcvt	(3)	output conversion
	ecvt	gcvt	(3)	output conversion
	end	edata	(3)	last locations in program
	end	end	(3)	last locations in program
	end	etext	(3)	last locations in program
	execl	execl	(3)	execute a file
	execl	execle	(3)	execute a file
	execl	exect	(3)	execute a file
	execl	execv	(3)	execute a file
	exit	exit	(3)	terminate a process after flushing any pending output
*	frexp	frexp	(3)	split into mantissa and exponent
*	frexp	ldexp	(3)	split into mantissa and exponent
*	frexp	modf	(3)	split into mantissa and exponent
	getenv	getenv	(3)	value for environment name
	getgrent	endgrent	(3)	get group file entry
	getgrent	getgrent	(3)	get group file entry
	getgrent	getgrgid	(3)	get group file entry
	getgrent	getgrnam	(3)	get group file entry
	getgrent	setgrent	(3)	get group file entry
	getlogin	getlogin	(3)	get a login name
	getpass	getpass	(3)	read a password
	getpwent	endpwent	(3)	get password file entry
	getpwent	getpwent	(3)	get password file entry
	getpwent	getpwnam	(3)	get password file entry
	getpwent	getpwuid	(3)	get password file entry
	getpwent	setpwent	(3)	get password file entry
†	getwd	getwd	(3)	get current working directory path name

*	ieee	classdouble	(3) IEEE arithmetic support functions
*	ieee	classfloat	(3) IEEE arithmetic support functions
*	ieee	copysign	(3) IEEE arithmetic support functions
*	ieee	drem	(3) IEEE arithmetic support functions
*	ieee	finite	(3) IEEE arithmetic support functions
*	ieee	isnan	(3) IEEE arithmetic support functions
*	ieee	logb	(3) IEEE arithmetic support functions
*	ieee	nextdouble	(3) IEEE arithmetic support functions
*	ieee	nextfloat	(3) IEEE arithmetic support functions
*	ieee	scalb	(3) IEEE arithmetic support functions
*	ieee	swapfpflag	(3) IEEE arithmetic support functions
*	ieee	swapfptrap	(3) IEEE arithmetic support functions
*	ieee	swapround	(3) IEEE arithmetic support functions
*	ieee	unordered	(3) IEEE arithmetic support functions
	insque	insque	(3) insert/remove element from a queue
	insque	remque	(3) insert/remove element from a queue
	intro	intro	(3) introduction to library functions
†	malloc	calloc	(3) memory allocator
†	malloc	free	(3) memory allocator
†	malloc	malloc	(3) memory allocator
†	malloc	realloc	(3) memory allocator
†	mktemp	mktemp	(3) make a unique file name
	monitor	moncontrol	(3) prepare execution profile
	monitor	monitor	(3) prepare execution profile
	monitor	monstartup	(3) prepare execution profile
†	nlist	nlist	(3) get entries from namelist
	perror	perror	(3) system error messages
	popen	pclose	(3) initiate I/O to/from a process
	popen	popen	(3) initiate I/O to/from a process
	psignal	psignal	(3) system signal messages
	qsort	qsort	(3) quicker sort
	random	initstate	(3) better random number generator; routines for changing generators
	random	random	(3) better random number generator; routines for changing generators
	random	setstate	(3) better random number generator; routines for changing generators
	random	srandom	(3) better random number generator; routines for changing generators
	regex	re_comp	(3) regular expression handler
	regex	re_exec	(3) regular expression handler
	scandir	alphasort	(3) scan a directory
	scandir	scandir	(3) scan a directory
	setjmp	_longjmp	(3) non-local goto
	setjmp	_setjmp	(3) non-local goto
	setjmp	longjmp	(3) non-local goto
	setjmp	setjmp	(3) non-local goto
	setuid	setegid	(3) set user and group ID
	setuid	seteuid	(3) set user and group ID
	setuid	setgid	(3) set user and group ID
	setuid	setrgid	(3) set user and group ID
	setuid	setruid	(3) set user and group ID
	setuid	setuid	(3) set user and group ID

sleep	sleep	(3) suspend execution for interval
string	index	(3) string operations
string	rindex	(3) string operations
string	strcat	(3) string operations
string	strcmp	(3) string operations
string	strcpy	(3) string operations
string	strlen	(3) string operations
string	strncat	(3) string operations
string	strncmp	(3) string operations
string	strncpy	(3) string operations
swab	swab	(3) swap bytes
syslog	closelog	(3) control system log
syslog	openlog	(3) control system log
syslog	syslog	(3) control system log
system	system	(3) issue a shell command
ttyname	isatty	(3) find name of a terminal
ttyname	ttyname	(3) find name of a terminal
ttyname	ttyslot	(3) find name of a terminal
valloc	valloc	(3) aligned memory allocator
varargs	va_arg	(3) variable argument list
varargs	va_end	(3) variable argument list
varargs	va_start	(3) variable argument list

1.4. Section 3F: FORTRAN Library

Man Page	Name	Section and Description
access	access	(3F) determine accessibility of a file
alarm	alarm	(3F) execute a subroutine after a specified time
bessel	besj0	(3F) functions of two kinds for integer orders
bessel	besj1	(3F) functions of two kinds for integer orders
bessel	besjn	(3F) functions of two kinds for integer orders
bessel	besy0	(3F) functions of two kinds for integer orders
bessel	besy1	(3F) functions of two kinds for integer orders
bessel	besyn	(3F) functions of two kinds for integer orders
bit	bit	(3F) and, or, xor, not, rshift, lshift, bitwise functions
chdir	chdir	(3F) change default directory
chmod	chmod	(3F) change mode of file
etime	dtime	(3F) return elapsed execution time
etime	etime	(3F) return elapsed execution time
exit	exit	(3F) terminate process with status
fdate	fdate	(3F) return date and time in an ASCII string
flmin	dflmax	(3F) return date and time in an ASCII string
flmin	dflmin	(3F) return date and time in an ASCII string
flmin	dffrac	(3F) return date and time in an ASCII string
flmin	ffrac	(3F) return date and time in an ASCII string
flmin	flmax	(3F) return date and time in an ASCII string
flmin	flmin	(3F) return extreme values
flmin	inmax	(3F) return extreme values
flush	flush	(3F) flush output to a logical unit
fork	fork	(3F) create a copy of this process
fseek	fseek	(3F) reposition a file on a logical unit
fseek	ftell	(3F) reposition a file on a logical unit

getarg	getarg	(3F)	return command line arguments
getarg	iargc	(3F)	return command line arguments
getc	getc	(3F)	get a character from a logical unit
getc	fgetc	(3F)	get a character from a logical unit
getcwd	getcwd	(3F)	get pathname of current working directory
getenv	getenv	(3F)	get value of environmental variables
getlog	getlog	(3F)	get user's login name
getlog	getlog0	(3F)	get user's login name
getpid	getpid	(3F)	get process id
getuid	getuid	(3F)	get user or group ID of the caller
getuid	getgid	(3F)	get user or group ID of the caller
hostnm	hostnm	(3F)	get name of current host
idate	idate	(3F)	return date or time in numerical form
idate	itime	(3F)	return date or time in numerical form
index	index	(3F)	tell about character objects
index	len	(3F)	tell about character objects
index	lnblnk	(3F)	tell about character objects
index	rindex	(3F)	tell about character objects
intro	intro	(3F)	introduction to FORTRAN library functions
ioinit	ioinit	(3F)	change f77 I/O initialization
kill	kill	(3F)	send a signal to a process
link	link	(3F)	make a link to an existing file
link	symlnk	(3F)	make a link to an existing file
loc	loc	(3F)	return the address of an object
long	long	(3F)	integer object conversion
long	short	(3F)	integer object conversion
perror	gerror	(3F)	get system error messages
perror	ierror	(3F)	get system error messages
perror	perrno	(3F)	get system error messages
putc	putc	(3F)	write a character to a FORTRAN logical unit
putc	fputc	(3F)	write a character to a FORTRAN logical unit
qsort	qsort	(3F)	quick sort
rand	drand	(3F)	return random values
rand	irand	(3F)	return random values
rand	irand	(3F)	return random values
range	dflmax	(3F)	return extreme values
range	dflmin	(3F)	return extreme values
range	flmax	(3F)	return extreme values
range	flmin	(3F)	return extreme values
range	inmax	(3F)	return extreme values
rename	rename	(3F)	rename a file
signal	signal	(3F)	change the action for a signal
sleep	sleep	(3F)	suspend execution for an interval
stat	fstat	(3F)	get file status
stat	lstat	(3F)	get file status
stat	stat	(3F)	get file status
system	system	(3F)	execute a UNIX command
time	ctime	(3F)	return system time
time	gmtime	(3F)	return system time
time	ltime	(3F)	return system time
time	time	(3F)	return system time
topen	tclose	(3F)	f77 tape I/O
topem	topen	(3F)	f77 tape I/O

	topen	tread	(3F) f77 tape I/O
	topen	trewin	(3F) f77 tape I/O
	topen	tskipf	(3F) f77 tape I/O
	topen	tstate	(3F) f77 tape I/O
	topen	twrite	(3F) f77 tape I/O
	traper	traper	(3F) trap arithmetic errors
	trapov	trapov	(3F) trap and repair floating point overflow
	trpfpe	fpecnt	(3F) trap and repair floating point faults
	trpfpe	trpfpe	(3F) trap and repair floating point faults
	ttynam	isaty	(3F) find name of a terminal port
	ttynam	ttynam	(3F) fine name of a terminal port
	unlink	unlink	(3F) remove a directory entry
	wait	wait	(3F) wait for a process to terminate

1.5. Section 3M: Math Library

	Man Page	Name	Section and Description
†	asinh	acosh	(3M) inverse hyperbolic functions
†	asinh	asinh	(3M) inverse hyperbolic functions
†	asinh	atanh	(3M) inverse hyperbolic functions
†	erf	erf	(3M) error functions
†	erf	erfc	(3M) error functions
	exp	exp	(3M) exponential, logarithm, power
	exp	expm1	(3M) exponential, logarithm, power
	exp	log	(3M) exponential, logarithm, power
	exp	log10	(3M) exponential, logarithm, power
	exp	log1p	(3M) exponential, logarithm, power
	exp	pow	(3M) exponential, logarithm, power
	exp	sqrt	(3M) exponential, logarithm, power
	floor	ceil	(3M) absolute value, floor, ceiling functions
	floor	fabs	(3M) absolute value, floor, ceiling functions
	floor	floor	(3M) absolute value, floor, ceiling functions
	gamma	lgamma	(3M) name changed to lgamma
	hypot	hypot	(3M) Euclidean distance
	hypot	cabs	(3M) Euclidean distance
	intro	intro	(3M) introduction to mathematical library functions
	j0	j0	(3M) Bessel functions
	j0	j1	(3M) Bessel functions
	j0	jn	(3M) Bessel functions
	j0	y0	(3M) Bessel functions
	j0	y1	(3M) Bessel functions
	j0	yn	(3M) Bessel functions
†	lgamma	lgamma	(3M) log gamma function
	sin	acos	(3M) trigonometric functions
	sin	asin	(3M) trigonometric functions
	sin	atan	(3M) trigonometric functions
	sin	atan2	(3M) trigonometric functions
	sin	cos	(3M) trigonometric functions
	sin	sin	(3M) trigonometric functions
	sin	tan	(3M) trigonometric functions
†	sinh	cosh	(3M) hyperbolic functions

†	sinh	sinh	(3M) hyperbolic functions
†	sinh	tanh	(3M) hyperbolic functions
†	sqrt	cbrt	(3M) cube root, square root
†	sqrt	sqrt	(3M) cube root, square root

1.6. Section 3N: Internet Network Library

Man Page	Name	Section and Description
byteorder	htonl	(3N) convert values between host and network byte order
byteorder	htons	(3N) convert values between host and network byte order
byteorder	ntohl	(3N) convert values between host and network byte order
byteorder	ntohs	(3N) convert values between host and network byte order
gethostent	endhostent	(3N) get network host entry
gethostent	gethostbyaddr	(3N) get network host entry
gethostent	gethostbyname	(3N) get network host entry
gethostent	gethostent	(3N) get network host entry
gethostent	sethostent	(3N) get network host entry
getnetent	endnetent	(3N) get network entry
getnetent	getnetbyaddr	(3N) get network entry
getnetent	getnetbyname	(3N) get network entry
getnetent	getnetent	(3N) get network entry
getnetent	setnetent	(3N) get network entry
getprotoent	endprotoent	(3N) get protocol entry
getprotoent	getprobyname	(3N) get protocol entry
getprotoent	getprobynumber	(3N) get protocol entry
getprotoent	getprotoent	(3N) get protocol entry
getprotoent	setprotoent	(3N) get protocol entry
getservent	endservent	(3N) get service entry
getservent	getservbyname	(3N) get service entry
getservent	getservbyport	(3N) get service entry
getservent	getservent	(3N) get service entry
getservent	setservent	(3N) get service entry
inet	inet_addr	(3N) internet address manipulation routines
inet	inet_lnaof	(3N) internet address manipulation routines
inet	inet_makeaddr	(3N) internet address manipulation routines
inet	inet_netof	(3N) internet address manipulation routines
inet	inet_network	(3N) internet address manipulation routines
inet	inet_ntoa	(3N) internet address manipulation routines
intro	intro	(3N) introduction to network library functions

1.7. Section 3S: C Standard I/O Library Subroutines

	Man Page	Name	Section and Description
	fclose	fclose	(3S) close or flush a stream
	fclose	fflush	(3S) close or flush a stream
†	ferror	clearerr	(3S) stream status inquiries
†	ferror	feof	(3S) stream status inquiries
†	ferror	ferror	(3S) stream status inquiries
†	ferror	fileno	(3S) stream status inquiries
†	fopen	fdopen	(3S) open a stream
†	fopen	fopen	(3S) open a stream
†	fopen	freopen	(3S) open a stream

	fread	fread	(3S) buffered binary input/output
	fread	fwrite	(3S) buffered binary input/output
†	fseek	fseek	(3S) reposition a stream
†	fseek	ftell	(3S) reposition a stream
†	fseek	rewind	(3S) reposition a stream
†	getc	fgetc	(3S) get character or word from stream
†	getc	getc	(3S) get character or word from stream
†	getc	getchar	(3S) get character or word from stream
†	getc	getw	(3S) get character or word from stream
†	gets	fgets	(3S) get a string from a stream
†	gets	gets	(3S) get a string from a stream
†	intro	stdio	(3S) standard buffered I/O package
	printf	fprintf	(3S) formatted output conversion
	printf	printf	(3S) formatted output conversion
	printf	sprintf	(3S) formatted output conversion
	putc	fputc	(3S) put character or word on a stream
	putc	putc	(3S) put character or word on a stream
	putc	putchar	(3S) put character or word on a stream
	putc	putw	(3S) put character or word on a stream
	puts	fputs	(3S) put a string on a stream
	puts	puts	(3S) put a string on a stream
	scanf	fscanf	(3S) formatted input conversion
	scanf	scanf	(3S) formatted input conversion
	scanf	sscanf	(3S) formatted input conversion
†	setbuf	setbuf	(3S) assign buffering to a stream
†	setbuf	setbuffer	(3S) assign buffering to a stream
†	setbuf	setlinebuf	(3S) assign buffering to a stream
	stdio	stdio	(3S) standard buffered input/output package
	ungetc	ungetc	(3S) push character back into input stream

1.8. Section 3X: Other Libraries

	Man Page	Name	Section and Description
	assert	assert	(3X) program verification
	curses	curses	(3X) screen functions with optimal cursor motion
	dbm	dbminit	(3X) data base subroutines
	dbm	delete	(3X) data base subroutines
	dbm	fetch	(3X) data base subroutines
	dbm	firstkey	(3X) data base subroutines
	dbm	nextkey	(3X) data base subroutines
	dbm	store	(3X) data base subroutines
*	fpa	fpa	(3X) direct interface to floating point accelerator
	getdiskbyname	getdiskbyname	(3X) get disk description by its name
	getfsent	endfsent	(3X) get file system descriptor file entry
	getfsent	getfsent	(3X) get file system descriptor file entry
	getfsent	getfsfile	(3X) get file system descriptor file entry
	getfsent	getfsspec	(3X) get file system descriptor file entry
	getfsent	getfstype	(3X) get file system descriptor file entry
	getfsent	setfsent	(3X) get file system descriptor file entry
	initgroups	initgroups	(3X) initialize group access list
	intro	intro	(3X) introduction to miscellaneous library functions
	plot	arc	(3X) graphics interface

plot	cont	(3X) graphics interface
plot	circle	(3X) graphics interface
plot	closepl	(3X) graphics interface
plot	erase	(3X) graphics interface
plot	label	(3X) graphics interface
plot	line	(3X) graphics interface
plot	linemod	(3X) graphics interface
plot	move	(3X) graphics interface
plot	openpl	(3X) graphics interface
plot	point	(3X) graphics interface
plot	space	(3X) graphics interface
rcmd	rcmd	(3X) routines for returning a stream to a remote command
rcmd	rresvport	(3X) routines for returning a stream to a remote command
rcmd	ruserok	(3X) routines for returning a stream to a remote command
rexec	rexec	(3X) return stream to a remote command
termcap	tgetent	(3X) terminal independent operation routines
termcap	tgetflag	(3X) terminal independent operation routines
termcap	tgetnum	(3X) terminal independent operation routines
termcap	tgetstr	(3X) terminal independent operation routines
termcap	tgoto	(3X) terminal independent operation routines
termcap	tputs	(3X) terminal independent operation routines

1.9. Section 3C: Compatibility Library Subroutines

Man Page	Name	Section and Description
alarm	alarm	(3C) schedule signal after specified time
getpw	getpw	(3C) get name from uid
intro	intro	(3C) introduction to compatibility library functions
nice	nice	(3C) set program priority
pause	pause	(3C) stop until signal
rand	rand	(3C) random number generator
rand	srand	(3C) random number generator
signal	signal	(3C) simplified software signal facilities
stty	gtty	(3C) set and get terminal state (defunct)
stty	stty	(3C) set and get terminal state (defunct)
time	ftime	(3C) get date and time
time	time	(3C) get date and time
times	times	(3C) get process times
utime	utime	(3C) set file times
vlimit	vlimit	(3C) control maximum system resource consumption
vtimes	vtimes	(3C) get information about resource utilization

1.10. Section 3G: Graphics Subroutines (located in Appendix C)

	Man Page	Name	Section and Description
*	circle	VI_Circle	(3G) draw a circle
*	clip	VI_Clip	(3G) set clipping window
*	color	VI_Color	(3G) change screen color
*	copy	VI_Copy	(3G) copy an area
*	cursor	VI_DisCur	(3G) control the display cursor
*	cursor	VI_EnCur	(3G) control the display cursor
*	cursor	VI_FDefnCur	(3G) control the display cursor

*	cursor	VI_MDefnCur	(3G) control the display cursor
*	cursor	VI_PosnCur	(3G) control the display cursor
*	dash	VI_Dash	(3G) set line dash pattern
*	font	VI_DropFont	(3G) select and manipulate fonts
*	font	VI_Font	(3G) select and manipulate fonts
*	font	VI_GetFont	(3G) select and manipulate fonts
*	force	VI_Force	(3G) force output of graphics orders
*	image	VI_FImage	(3G) draw an image
*	image	VI_MImage	(3G) draw an image
*	init	VI_Init	(3G) initialize and terminate the subroutine interface
*	init	VI_Term	(3G) initialize and terminate the subroutine interface
*	intro	intro	(3G) introduction to display graphics subroutines
*	log	VI_Login	(3G) begin logging subroutine calls and close a log file
*	log	VI_Logout	(3G) begin logging subroutine calls and close a log file
*	line	VI_ALine	(3G) draw a line
*	line	VI_RLine	(3G) draw a line
*	move	VI_AMove	(3G) move the current point
*	move	VI_RMove	(3G) move the current point
*	merge	VI_Merge	(3G) set merge mode
*	query	VI_QClip	(3G) query graphics parameters
*	query	VI_QColor	(3G) query graphics parameters
*	query	VI_QDash	(3G) query graphics parameters
*	query	VI_QFont	(3G) query graphics parameters
*	query	VI_QMerge	(3G) query graphics parameters
*	query	VI_QPoint	(3G) query graphics parameters
*	query	VI_QWidth	(3G) query graphics parameters
*	read	VI_FRead	(3G) read display data
*	read	VI_MRead	(3G) read display data
*	run	VI_Run	(3G) process a log file
*	string	VI_String	(3G) draw a string
*	tile	VI_Tile	(3G) tile a rectangle
*	width	VI_Width	(3G) set line width

1.11. Section 4: Special Files

	Man Page	Name	Section and Description
*	aedemul	aedemul	(4) graphics interfaces for the IBM Academic Information Systems experimental display
*	ap	ap	(4) asychronous data mode protocol line discipline
	arp	arp	(4P) Address Resolution Protocol
*	asy	asy	(4) multi-port asynchronous communications RS232C interface
	autoconf	autoconf	(4) diagnostics from the autoconfiguration code
	bk	bk	(4) line discipline for machine-machine communication
*	bufemul	bufemu	(4) kernel buffering emulator
*	bus	bus	(4) control of access to the system I/O bus
	cons	cons	(4) keyboard and monochrome screen console interface
*	disk	disk	(4) format of reserved areas of the hard disk
	drum	drum	(4) paging device
*	fd	fd	(4) diskette interface
*	hd	hd	(4) PC/AT hard disk controller interface
*	ibm5151	ibm5151	(4) IBM 5151 Monochrome Display interface

*	ibm5151	mono	(4) IBM 5151 Monochrome Display interface
*	ibm6153	apa8	(4) IBM 6153 Advanced Monochrome Graphics Display interface
*	ibm6153	ibm6153	(4) IBM 6153 Advanced Monochrome Graphics Display interface
*	ibm6154	apa8c	(4) IBM 6154 Advanced Color Graphics Display interface
*	ibm6154	ibm6154	(4) IBM 6154 Advanced Color Graphics Display interface
*	ibm6155	apa16	(4) IBM 6155 Extended Monochrome Graphics Display interface
*	ibm6155	ibm6155	(4) IBM 6155 Extended Monochrome Graphics Display interface
*	ibmaed	ibmaed	(4) IBM Academic Information Systems experimental display
*	ibmemul	ibmemul	(4) IBM 3101 emulator
	imp	imp	(4P) IMP raw socket interface
	inet	inet	(4F) Internet protocol family
	intro	intro	(4) introduction to special files and hardware support
	intro	networking	(4N) introduction to networking facilities
	ip	ip	(4P) Internet Protocol
*	kbdemul	kbdemul	(4) default keyboard emulator
*	lan	lan	(4) IBM 6100 Token-Ring Network Adapter
	lo	lo	(4) software loopback network interface
	lp	lp	(4) line printer
	mem	kmem	(4) main memory
	mem	mem	(4) main memory
*	mouse	mouse	(4) mouse interface
	mtio	mtio	(4) UNIX magtape interface
	null	null	(4) data sink
*	psp	psp	(4) planar serial port RS232C interface
	pty	pty	(4) pseudo terminal driver
	pup	pup	(4F) Xerox PUP-I protocol family
	pup	pup	(4P) raw PUP socket interface
*	rvd	rvd	(4P) Remote Virtual Disk protocol
*	speaker	speaker	(4) console speaker interface
*	st	st	(4) streaming-tape interface
*	stdemul	stdemul	(4) standard output emulator
†	tb	tb	(4) line discipline for digitizing devices
	tcp	tcp	(4P) Internet Transmission Control Protocol
	tty	tty	(4) general terminal interface
	udp	udp	(4P) Internet User Datagram Protocol
	un	un	(4) IBM RT PC Baseband Adapter for use with Ethernet
*	xemul	xemul	(4) X input emulator for queuing keyboard and mouse events

1.12. Section 5: File Formats

Man Page	Name	Section and Description
a.out	a.out	(5) assembler and link editor output
acct	acct	(5) execution accounting file
aliases	aliases	(5) aliases file for sendmail
ar	ar	(5) archive (library) file format

*	consoles	consoles	(5) utility data base of display screens
	core	core	(5) format of memory image file
	dir	dir	(5) format of directories
	disktab	disktab	(5) disk description file
	dump	dump	(5) incremental dump format
	dump	dumpdates	(5) incremental dump format
*	font3812	font3812	(5) font structures for the 3812 fonts
	fs	fs	(5) format of file system volume
	fs	inode	(5) format of file system volume
	fstab	fstab	(5) static information about the file systems
	gettytab	gettytab	(5) terminal configuration data base
	group	group	(5) group file
	hosts	hosts	(5) host name data base
*	keyboard_codes	keyboard_codes	(5) keyboard scancode table
†	map3270	map3270	(5) data base for mapping ASCII keystrokes into IBM 3270 displays
	mtab	mtab	(5) mounted file system table
	networks	networks	(5) network name data base
	passwd	passwd	(5) password file
	phones	phones	(5) remote host phone number data base
	plot	plot	(5) graphics interface
	printcap	printcap	(5) printer capability data base
*	printer3812	printer3812	(5) IBM 3812 Pageprinter status information
	protocols	protocols	(5) protocol name data base
	remote	remote	(5) remote host description file
*	rvddb	rvddb	(5) Remote Virtual Disk (RVD) server configuration table
*	rvdtab	rvdtab	(5) information about client Remote Virtual Disks (RVDs)
	services	services	(5) service name data base
	stab	stab	(5) symbol table types
	tar	tar	(5) tape archive file format
	termcap	termcap	(5) terminal capability data base
	ttys	ttys	(5) terminal initialization data
	ttytype	ttytype	(5) data base of terminal types by port
	types	types	(5) primitive system data types
	utmp	utmp	(5) login records
	utmp	wtmp	(5) login records
	uuencode	uuencode	(5) format of an encoded uuencode file
	vfont	vfont	(5) font formats for the Benson-Varian or Versatec

1.13. Section 6: Games

NONE

1.14. Section 7: Miscellaneous

Man Page	Name	Section and Description
ascii	ascii	(7) map of ASCII character set
environ	environ	(7) user environment
eqnchar	eqnchar	(7) special character definitions for eqn
hier	hier	(7) file system hierarchy
intro	miscellaneous	(7) miscellaneous useful information pages
mailaddr	mailaddr	(7) mail addressing description
man	man	(7) macros to typeset manual

me	me	(7) macros for formatting papers
ms	ms	(7) text formatting macros
term	term	(7) conventional names for terminals

1.15. Section 8: System Maintenance

	Man Page	Name	Section and Description
	ac	ac	(8) login accounting
	adduser	adduser	(8) procedure for adding new users
*	aedtest	aedtest	(8) IBM Academic Information Systems experimental display self-tests
	analyze	analyze	(8) virtual UNIX postmortem crash analyzer
	arcv	arcv	(8) convert archives to new format
	badsect	badsect	(8) create files to contain bad sectors
	bugfiler	bugfiler	(8) file bug reports in folders automatically
	catman	catman	(8) create the cat files for the manual
	chown	chown	(8) change owner
	clri	clri	(8) clear i-node
	comsat	comsat	(8C) biff server
	config	config	(8) build system configuration files
	crash	crash	(8R) what happens when the system crashes
	cron	cron	(8) clock daemon
*	cvt3812	cvt3812	(8) convert IBM 3820 and IBM 3800 fonts for use with the IBM 3812 Pageprinter
*	cvt3812	cvt20to12	(8) convert IBM 3820 and IBM 3800 fonts for use with the IBM 3812 Pageprinter
*	cvt3812	cvt00to12	(8) convert IBM 3820 and IBM 3800 fonts for use with the IBM 3812 Pageprinter
*	cvtsym	cvtsym	(8) convert symbol table
	dcheck	dcheck	(8) file system directory consistency check
*	debug	debug	(8) debugger for the IBM RT PC
*	diskpart	diskpart	(8) calculate default disk partition sizes
	dmesg	dmesg	(8) collect system diagnostic messages to form error log
	drtest	drtest	(8) standalone disk test program
	dump	dump	(8) incremental file system dump
	dumpfs	dumpfs	(8) dump file system information
	edquota	edquota	(8) edit user quotas
	fastboot	fastboot	(8) reboot/halt the system without checking the disks
	fastboot	fasthalt	(8) reboot/halt the system without checking the disks
*	fdformat	fdformat	(8R) format diskettes
*	flcopy	flcopy	(8R) copier for diskettes
	format	format	(8R) format hard disks
	fsck	fsck	(8) file system consistency check and interactive repair
	ftpd	ftpd	(8C) DARPA Internet File Transfer Protocol daemon
	gettable	gettable	(8C) get NIC-format host tables from a host
	getty	getty	(8) set terminal mode
	halt	halt	(8) stop the processor
	htable	htable	(8) get NIC-format host tables from a host
*	ibm3812pp	ibm3812pp	(8) IBM 3812 Pageprinter server
	icheck	icheck	(8) file system storage consistency check
	ifconfig	ifconfig	(8C) configure network interface parameters
	init	init	(8) process control initialization

	intro	intro	(8) introduction to system maintenance and operation commands
	kgmon	kgmon	(8) generate a dump of the operating system's profile buffers
*	landump	landump	(8R) dump IBM Token-Ring Personal Computer Adapter
	lpc	lpc	(8) line printer control program
	lpd	lpd	(8) line printer daemon
*	lpfilter	ibmbit	(8R) output filters for the IBM 4201 Proprinter and IBM 5152 Graphics Printer
*	lpfilter	ibmgra	(8R) output filters for the IBM 4201 Proprinter and IBM 5152 Graphics Printer
*	lpfilter	ibmpro	(8R) output filters for the IBM 4201 Proprinter and IBM 5152 Graphics Printer
	makedev	makedev	(8) make system special files
	makekey	makekey	(8) generate encryption key
*	makesym	makesym	(8) make debugger symbol table
*	minidisk	minidsk	(8R) minidisk maintenance utility
	mkfs	mkfs	(8) construct a file system
	mklost+found	mklost+found	(8) make a lost+found directory for fsck
	mknod	mknod	(8) build special file
	mkproto	mkproto	(8) construct a prototype file system
	mount	mount	(8) mount and dismount file system
	mount	umount	(8) mount and dismount file system
	ncheck	ncheck	(8) generate names from i-numbers
	newfs	newfs	(8) construct a new file system
*	omerge	omerge	(8) merge object files
	pac	pac	(8) printer/plotter accounting information
*	ppt	ppt	(8) text filter for the IBM 3812 Pageprinter
	pstat	pstat	(8) print system facts
	quot	quot	(8) summarize file system ownership
	quotacheck	quotacheck	(8) file system quota consistency checker
	quotaon	quotaoff	(8) turn file system quotas on and off
	quotaon	quotaon	(8) turn file system quotas on and off
	rc	rc	(8) command script for auto-reboot and daemons
	rdump	rdump	(8C) file system dump across the network
	reboot	reboot	(8) UNIX bootstrapping procedures
	renice	renice	(8) alter priority of running processes
	repquota	repquota	(8) summarize quotas for a file system
	restore	restore	(8) incremental file system restore
	rexecd	rexecd	(8C) remote execution daemon
	rlogind	rlogind	(8C) remote login daemon
	rmt	rmt	(8C) remote magtape protocol module
	route	route	(8C) manually manipulate the routing tables
	routed	routed	(8C) network routing daemon
	rrestore	rrestore	(8C) restore a file system dump across the network
	rshd	rshd	(8C) remote shell daemon
*	rvdchlog	rvdchlog	(8) change logging level of Remote Virtual Disk (RVD) server
*	rvddown	rvddown	(8) force spindown of a Remote Virtual Disk (RVD) pack
*	rvdexch	rvdexch	(8) exchange names of two Remote Virtual Disk (RVD) packs
*	rvdflush	rvdflush	(8) spindown client's Remote Virtual Disk (RVD) packs

*	rvdgetm	rvdgetm	(8) get operations message from Remote Virtual Disk (RVD) server
*	rvdlog	rvdlog	(8) cause Remote Virtual Disk (RVD) server to log statistics
*	rvdsend	rvdsend	(8) send control stream to Remote Virtual Disk (RVD) server
*	rvdsetm	rvdsetm	(8) set operations message on Remote Virtual Disk (RVD) server
*	rvdshow	rvdshow	(8) show connections to Remote Virtual Disk (RVD) server
*	rvdshut	rvdshut	(8) force shutdown of Remote Virtual Disk (RVD) server
*	rvdsrv	rvdsrv	(8) Remote Virtual Disk (RVD) server daemon
	rwhod	rwhod	(8C) system status daemon
	sa	accton	(8) system accounting
	sa	sa	(8) system accounting
*	sautil	sautil	(8R) standalone utility package
	savecore	savecore	(8) save a core dump of the operating system
*	savervd	savephys	(8) back up and restore Remote Virtual Disk (RVD) packs to and from tape
*	savervd	savervd	(8) back up and restore Remote Virtual Disk (RVD) packs to and from tape
*	savervd	zaprvd	(8) back up and restore Remote Virtual Disk (RVD) packs to and from tape
	sendmail	sendmail	(8) send mail over the Internet
*	setscreen	setscreen	(8) control display screen access
	shutdown	shutdown	(8) close down the system at a given time
*	spinup	spindown	(8) spin up/down Remote Virtual Disk (RVD) pack
*	spinup	spinup	(8) spin up/down Remote Virtual Disk (RVD) pack
	sticky	sticky	(8) executable files with persistent text
	swapon	swapon	(8) specify additional device for paging and swapping
	sync	sync	(8) update the super block
	syslog	syslog	(8) log systems messages
	telnetd	telnetd	(8C) DARPA TELNET protocol daemon
	tftpd	tftpd	(8C) DARPA Trivial File Transfer Protocol server
	trpt	trpt	(8C) transliterate protocol trace
	tunefs	tunefs	(8) tune up an existing file system
	update	update	(8) periodically update the superblock
	uuclean	uuclean	(8C) uucp spool directory clean-up
	uusnap	uusnap	(8C) show snapshot of the UUCP system
*	vddb	vddb	(8) Remote Virtual Disk (RVD) data base manager
*	vdstats	vdstats	(8) list client Remote Virtual Disk (RVD) statistics
	vipw	vipw	(8) edit the password file
*	width3812	width3812	(8) build width tables for the IBM 3812 Pageprinter

2. UNSUPPORTED FUNCTIONS

The following sections list the functions of 4.2BSD for the VAX which are not supported by 4.2/RT.

2.1. Section 1: Commands and Application Programs.

Man Page	Name	Section and Description
fed	fed	(1) font editor
fp	fp	(1) functional programming language compiler/interpreter
lisp	lisp	(1) Lisp interpreter
liszt	liszt	(1) compile a Franz Lisp programx
lxref	lxref	(1) Lisp cross reference program
pc	pc	(1) Pascal compiler
pdx	pdx	(1) Pascal debugger
pi	pi	(1) Pascal interpreter code translator
pix	pix	(1) Pascal interpreter and executor
pmerge	pmerge	(1) Pascal file merger
px	px	(1) Pascal interpreter
pxp	pxp	(1) Pascal execution profiler
pxref	pxref	(1) Pascal cross-reference program
tc	tc	(1) phototypesetter simulator
tp	tp	(1) manipulate tape archive
tk	tk	(1) paginator for the Tektronix 4014
vfontinfo	vfontinfo	(1) inspect and print out information about UNIX fonts
vlp	vlp	(1) format Lisp programs to be printed with nroff, vtroff, or troff
vpr	vpq	(1) raster printer/plotter spooler
vpr	vpr	(1) raster printer/plotter spooler
vpr	vprint	(1) raster printer/plotter spooler
vpr	vprm	(1) raster printer/plotter spooler
vtroff	vtroff	(1) troff to a raster plotter
vwidth	vwidth	(1) make troff width table for a font

2.2. Section 2: System Calls

NONE

2.3. Section 3: C Library Subroutines

NONE

2.4. Section 3F: FORTRAN Library

NONE

2.5. Section 3G: AED Graphics Subroutines

NONE

2.6. Section 3M: Math Library

NONE

2.7. Section 3N: Internet Network Library

NONE

2.8. Section 3S: C Standard I/O Library Subroutines

NONE

2.9. Section 3X: Other Libraries

Man Page	Name	Section and Description
lib2648	lib2648	(3X) subroutines for the HP 2648 graphics terminal

2.10. Section 3C: Compatibility Library Subroutines

NONE

2.11. Section 4: Special Files

Man Page	Name	Section and Description
acc	acc	(4) ACC LH/DH IMP interface
ad	ad	(4) Data Translation A/D converter
css	css	(4) DEC IMP-11A LH/DH IMP interface
ct	ct	(4) phototypesetter interface
dh	dh	(4) DH-11/DM-11 communications multiplexer
dmc	dmc	(4) DEC DMC-11/DMR-11 point-to-point communications device
dmf	dmf	(4) DMF-32, terminal multiplexer
dn	dn	(4) DN-11 autocall unit interface
dz	dz	(4) DZ-11 communications multiplexer
ec	ec	(4) 3Com 10 Mb/s Ethernet interface
en	en	(4) Xerox 3 Mb/s Ethernet interface
fl	fl	(4) console diskette interface
hk	hk	(4) RK6-11/RK06 and RK07 moving head disk
hp	hp	(4) MASSBUS disk interface
ht	ht	(4) TM-03/TE-16,TU-45,TU-77 MASSBUS magtape interface
hy	hy	(4) Network Systems Hyperchannel interface
ik	ik	(4) Ikonas frame buffer, graphics device interface
il	il	(4) Interlan 10 Mb/s Ethernet interface
imp	imp	(4) 1822 network interface
kg	kg	(4) KL-11/DL-11W line clock
mt	mt	(4) TM78/TU-78 MASSBUS magtape interface
pcl	pcl	(4) DEC CSS PCL-11 B Network Interface
ps	ps	(4) Evans and Sutherland Picture System 2 graphics device interface
rx	rx	(4) DEC RX02 diskette interface
tm	tm	(4) TM-11/TE-10 magtape interface
ts	ts	(4) TS-11 magtape interface
tu	tu	(4) /730 and VAX-11/750 TU58 console cassette interface
uda	uda	(4) UDA-50 disk controller interface
up	up	(4) unibus storage module controller/drives
ut	ut	(4) UNIBUS TU45 tri-density tape drive interface
uu	uu	(4) TU58/DECtape II UNIBUS cassette interface
va	va	(4) Benson-Varian interface
vp	vp	(4) Versatec interface
vv	vv	(4) Proteon proNET 10 Megabit ring

2.12. Section 5: File Formats

Man Page	Name	Section and Description
tp	tp	(5) DEC/mag tape formats

vfont	vfont	(5) font formats for the Benson-Varian or Versatec
vgrindefs	vgrindefs	(5) vgrind's language definition data base

2.13. Section 6: Games

Man Page	Name	Section and Description
aardvark	aardvark	(6) yet another exploration game
adventure	adventure	(6) an exploration game
arithmetic	arithmetic	(6) provide drill in number facts
backgammon	backgammon	(6) the game of backgammon
banner	banner	(6) print large banner on printer
bcd	bcd	(6) convert to antique media
boggle	boggle	(6) the game of boggle
canfield	canfield	(6) the solitaire card game Canfield
canfield	cfscores	(6) the solitaire card game Canfield
chess	chess	(6) the game of chess
ching	ching	(6) the book of changes and other cookies
cribbage	cribbage	(6) the card game cribbage
doctor	doctor	(6) interact with a psychoanalyst
fish	fish	(6) play Go Fish
fortune	fortune	(6) print a random, hopefully interesting, adage
hangman	hangman	(6) computer version of the game hangman
mille	mille	(6) play Mille Bournes
monop	monop	(6) the game of Monopoly
number	number	(6) convert Arabic numerals to English
quiz	quiz	(6) test your knowledge
rain	rain	(6) animated raindrops display
rogue	rogue	(6) exploring the dungeons of doom
snake	snake	(6) display chase game
snake	snscore	(6) display chase game
trek	trek	(6) trekkie game
worm	worm	(6) the growing worm game
worms	worms	(6) animate worms on a display terminal
wump	wump	(6) the game of hunt-the-wumpus

2.14. Section 7: Miscellaneous

NONE

2.15. Section 8: System Maintenance

Man Page	Name	Section and Description
arff	arff	(8R) archiver and copier for diskette
arff	flcopy	(8) archiver and copier for diskette
bad144	bad144	(8) read/write DEC standard 144 bad sector information
implog	implog	(8C) IMP log interpreter
implogd	implogd	(8C) IMP logger process
rxformat	rxformat	(8V) format diskettes

This page intentionally left blank.

Appendix B. Graphics Manual Pages for the IBM Academic Information Systems Experimental Display

This section contains the manual pages for section 3G; they describe the display graphics subroutines. You may want to file these manual pages in Volume I.

- intro (3G)
- circle (3G)
- clip (3G)
- color (3G)
- copy (3G)
- cursor (3G)
- dash (3G)
- font (3G)
- force (3G)
- image (3G)
- init (3G)
- line (3G)
- log (3G)
- merge (3G)
- move (3G)
- query (3G)
- read (3G)
- run (3G)
- string (3G)
- tile (3G)
- width (3G)

This page intentionally left blank.

NAME

VI_Circle − draw a circle

SYNOPSIS

```
VI_Circle(radius)
     int radius;          /* circle radius */
```

DESCRIPTION

VI_Circle draws a circle with the specified radius and the current point as its center. The current point is unchanged.

NOTE

VI_Circle applies only to the IBM Academic Information Systems experimental display. The line attributes *VI_Dash* and *VI_Width* do not apply to *VI_Circle.*

Nothing is drawn if the radius is less than or equal to zero. You cannot use concentric circles to do a solid area fill.

This page intentionally left blank.

NAME

VI_Clip – set clipping window

SYNOPSIS

```
VI_Clip(lx,ly,hx,hy)
        int lx,ly;        /* top left corner of clipping area */
        int hx,hy;        /* bottom right corner of clipping area */
```

DESCRIPTION

VI_Clip specifies that subsequent primitives drawn on the screen are to be clipped to the specified area. It is the user's responsibility to ensure the sensibility of the window definition. The clipping window is initially set to the whole screen.

NOTE

VI_Clip applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

query(3G)

This page intentionally left blank.

NAME

VI_Color − change screen color

SYNOPSIS

```
VI_Color(color)
        int color;              /* new color, true for white */
```

DESCRIPTION

VI_Color sets the color of the screen to the specified value: 0 means that bits having the binary value "0" will be black on the screen; 1 means that bits having the binary value "1" will be black on the screen. If this value is different from the previous value, the screen will be inverted, so as to make the change transparent to the application. The screen color is initially white 1's on black 0's, color 0.

NOTE

VI_Color applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

query(3G)

This page intentionally left blank.

NAME

VI_Copy − copy an area

SYNOPSIS

```
VI_Copy(sx,sy,tx,ty,wd,ht,merge)
         int sx,sy;              /* source top-left */
         int tx,ty;              /* target top-left */
         int wd,ht;        /* rectangle dimensions */
         int merge;        /* merge mode */
```

DESCRIPTION

VI_Copy duplicates the rectangle at *sx,sy* with the dimensions *wd,ht* to the point *tx,ty*. The copied bits are merged with the target area using the specified merge mode, not the merge mode set by *merge*(3G). See *merge*(3G) for a description of merge modes.

Both the source and destination rectangles must be completely on the screen. The current setting of the clipping window is ignored.

NOTE

VI_Copy applies only to the IBM Academic Information Systems experimental display.

VI_Copy cannot copy an area onto itself with a mode change, e.g. for highlighting. A fast way to highlight is to use *VI_Merge* with XOR mode and *VI_Tile*.

SEE ALSO

merge (3G)

This page intentionally left blank.

NAME

VI_MDefnCur, VI_FDefnCur, VI_EnCur, VI_DisCur, VI_PosnCur – control the display cursor

SYNOPSIS

```
VI_MDefnCur(xoff,yoff,black,white)
        int xoff;              /* x offset of cursor center */
        int yoff;              /* y offset of cursor center */
        unsigned short *black; /* first byte of black mask */
        unsigned short *white; /* first byte of white mask */

VI_FDefnCur(filename)
        char *filename;        /* name of cursor definition file */

VI_EnCur()

VI_DisCur()

VI_PosnCur(x,y)
        int x,y;                     /* new cursor position */
```

DESCRIPTION

These subroutines allow programs to control the display cursor by defining it, enabling and disabling it, and changing its position. Disabling and reenabling the cursor do not affect its pattern or position. Because the display maintains the cursor separately from the display buffer, the cursor does not have to be removed when a graphics primitive intersects its position. Initially the cursor is transparent and disabled, and is positioned at the center of the screen.

VI_MDefnCur Sets the cursor as specified. *xoff,yoff* is the displacement of the cursor pattern from the current position of the cursor. For example, a value of (32,32) would center the cursor pattern around the current point. The cursor pattern itself is a 64-by-64 bit image, with two planes. A 1 in the black plane indicates that that bit of the cursor should be black. A 1 in the white plane indicates that the cursor should be white in that position. If a bit has a 0 in both planes, the cursor is transparent in that position. If a bit is 1 in both planes, the cursor is white. The two planes are images in the same format as accepted by *MImage,* and must be 64-by-64, or 512 bytes each.

VI_FDefnCur Sets the cursor to the definition in the specified file. The file has the format shown below; the fields are explained under *MDefnCur*.

Offset (bytes)	Description
0	XOFF
2	YOFF
4	BLACK bit pattern
516	WHITE bit pattern

See the description of *MDefnCur* for a description of the fields.

VI_EnCur Enables the cursor and displays it if it is not already present.

VI_DisCur Disables the cursor and removes it from the screen if it is present.

VI_PosnCur Moves the cursor to the specified position. It cannot be moved off the screen.

NOTE

VI_Cursor applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

image(3G)

This page intentionally left blank.

NAME

VI_Dash – set line dash pattern

SYNOPSIS

```
VI_Dash(dash,dashlen)
        unsigned short dash;          /* dash pattern */
        short dashlen;                /* dash pattern length */
```

DESCRIPTION

If no dash pattern has been set, lines drawn with the *VI_RLine* and *VI_ALine* subroutines described under *line*(3G) are solid lines of 1's. If a pattern has been set, the bits of the pattern word are used in sequence whenever the vector generator would normally output a 1. Setting a pattern of 0x5555 produces a very acceptable dotted line. Other patterns may be used to vary the size of dashes in the line. The length of the pattern can range from 1 to 16 bits. The pattern bits should be left-justified. Setting the pattern length to 0 specifies a return to solid lines. The line dash pattern is initially set to solid 1's.

SEE ALSO

line(3G), merge(3G), query(3G), width(3G)

NOTE

VI_Dash applies only to the IBM Academic Information Systems experimental display. *VI_Dash* does not support *VI_Circle*.

This page intentionally left blank.

NAME

VI_Font, VI_GetFont, VI_DropFont – select and manipulate fonts

SYNOPSIS

```
VI_Font(fontid)
        int fontid;            /* font ID */

VI_GetFont(name,fontid)
        char *name;            /* font name */
        short *fontid;         /* font ID */

VI_DropFont(fontid)
        int fontid;            /* ID of font to release */
```

DESCRIPTION

Fonts are stored in files, which are loaded into the workstation memory when requested by applications using *VI_GetFont*. Once a font is loaded, it is kept in memory until the program ends, unless explicitly dropped with *VI_DropFont*.

VI_GetFont	Loads the specified font into memory, if it is not already present. If the font is successfully loaded, the font ID is returned. Setting the current font to this ID with *VI_Font* causes subsequent strings to be displayed in the font.
VI_Font	Selects the font with the specified font ID. Font IDs range from 0 to 255 and are returned by calls to *VI_GetFont*.
VI_DropFont	Drops the specified font from memory. The application should not attempt to use the font ID again. If the font is required, a new font ID should be generated by a request to *VI_GetFont*.

NOTE

VI_Font applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

string(3G)

DIAGNOSTICS

If *VI_GetFont* returns a font ID of 0, either the font could not be found, or it did not fit in memory. If the font did not fit in memory, a message will be sent to *stderr*.

This page intentionally left blank.

NAME

VI_Force − force output of graphics orders

SYNOPSIS

VI_Force()

DESCRIPTION

Commands built with subroutines described in "Setting Graphics Parameters" and "Issuing Graphics Primitives" in "The C Subroutine Interface for the IBM Academic Information Systems Experimental Display" generally do not send their output to the screen immediately. Instead the output remains in a buffer until the buffer is full, when its output is sent to the screen. Use *VI_Force* to force output in the current buffer to be transmitted before the buffer is full.

NOTE

VI_Force applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

init(3G)

This page intentionally left blank.

NAME

VI_MImage, VI_FImage – draw an image

SYNOPSIS

```
VI_MImage(wd,ht,data)
        int wd,ht;                  /* dimensions of image */
        unsigned short *data;       /* first byte of image */

VI_FImage(filename)
        char *filename;          /* file name of image to draw */
```

DESCRIPTION

These functions draw an image from memory or from a file. The current point is unchanged. The image data should be in scanline order, from top to bottom, with each scanline padded to the next 16-bit word. For example, for a width of WD and height of HT, there should be 2*HT(WD + 15)/16 bytes of image data.

VI_MImage Draws an image of the specified dimensions whose top left corner is at the current point. *data* must be the first byte of an image large enough to fill the rectangle specified by *wd* and *ht,* or an addressing error may result.

VI_FImage Draws the image contained in the specified file, placing its top left corner at the current point. The image file must have the following format:

Offset (bytes)	Description
0	The width of the image
2	The height of the image
4	Image data

NOTE

VI_Image applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

read(3G)

This page intentionally left blank.

NAME

VI_Init, VI_Term – initialize and terminate the subroutine interface

SYNOPSIS

```
VI_Init(wd,ht)
        short *wd,*ht;          /* screen dimensions */

VI_Term()
```

DESCRIPTION

These functions initialize and terminate the subroutine interface.

VI_Init	Initializes the display and returns the dimensions of the screen. The display currently has a width of 1024 bits and a height of 800 bits. *VI_Init* must be the first call. The top left point is (0,0); the bottom right point is (1023,799).
VI_Term	Completes processing, closes any log files, and forces transmission of the graphics buffer to the display.

FILES

/dev/aed
/usr/lib/aed/whim.aed
/usr/lib/aed/pcfont.fnt

NOTE

VI_Init applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

force(3G), log(3G)

This page intentionally left blank.

NAME

intro − introduction to display graphics subroutines

DESCRIPTION

This section describes the subroutines that are part of the interface for the IBM Academic Information Systems experimental display (herein after called "the experimental display"). The subroutines are graphics routines for controlling the experimental display in all-points addressable mode.

The interface described in this section provides access to a set of functions designed to support a window manager, and is composed primarily of subroutines, as distinguished from functions. A typical subroutine uses parameters to receive input and return output. C passes parameters by value; to call a subroutine which returns information, you must supply an address for the returning value as the parameter.

Calls that supply an address for return in this package should usually supply the address of a *short* (16-bit) integer. Calls that pass integer values can usually get by with either *short* or *int*. See the individual routines.

Many of the subroutines do return a value as a function would, generally for error return codes and special case handling. It is strongly recommended that applications monitor return codes to prevent bizarre events and possibly more severe errors. When linking, specify *-laed* to pick up the experimental-display library.

All subroutines use screen coordinates with the origin in the upper left corner of the experimental display.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
VI_ALine	line.3g	draw a line to an absolute location
VI_AMove	move.3g	move the current point to an absolute location
VI_Circle	circle.3g	draw a circle
VI_Clip	clip.3g	set clipping window
VI_Color	color.3g	change screen color
VI_Copy	copy.3g	copy an area
VI_Dash	dash.3g	set line dash pattern
VI_DisCur	cursor.3g	disable cursor
VI_DropFont	font.3g	release font
VI_EnCur	cursor.3g	enable cursor
VI_FDefnCur	cursor.3g	set cursor pattern from file
VI_FImage	image.3g	draw an image from a file
VI_Font	font.3g	select font
VI_Force	force.3g	force output of graphics orders
VI_Fread	read.3g	read experimental-display data into a file
VI_GetFont	font.3g	load a font into memory
VI_Init	init.3g	initialize the subroutine interface
VI_Login	log.3g	begin logging subroutine calls
VI_Logout	log.3g	close a log file
VI_MDefnCur	cursor.3g	set cursor pattern from memory
VI_Merge	merge.3	set merge mode
VI_MImage	image.3g	draw an image from memory
VI_MRead	read.3g	read experimental-display data into memory
VI_PosnCur	cursor.3g	set cursor position
VI_QClip	query.3	query clipping rectangle
VI_QColor	query.3g	query current color
VI_QDash	query.3g	query dash pattern

VI_QFont	query.3g	query font
VI_QMerge	query.3g	query merge mode
VI_QPoint	query.3g	query current point
VI_QWidth	query.3g	query line width
VI_RLine	line.3g	draw a line to a relative location
VI_RMove	move.3g	move the current point to a relative location
VI_Run	run.3g	process a log file
VI_String	string.3g	draw a string
VI_Term	init.3g	terminate the subroutine interface
VI_Tile	tile.3g	tile a rectangle
VI_Width	width.3g	set line width

FILES

/usr/lib/aed/whim.aed
/usr/lib/aed/pcfont.fnt
/usr/lib/libaed.a
/usr/src/usr.lib/libaed/examples
/dev/aed

NOTE

These subroutines apply only to the IBM Academic Information Systems experimental display.

SEE ALSO

"The C Subroutine Interface for the IBM Academic Information Systems Experimental Display" in Volume II, Supplementary Documents.

NAME

VI_ALine, VI_RLine − draw a line

SYNOPSIS

```
VI_ALine(x,y)
        int x,y;        /* end point of line */

VI_RLine(dx,dy)
        int dx,dy;      /* displacement to end point */
```

DESCRIPTION

These functions draw a line to an absolute or a relative location. A line is normally of 1's, and is merged with the window data according to the current merge mode.

VI_ALine Draws a line from the current point to the specified point (the line's end point) according to the current values of the merge, width, and dash pattern parameters. The specified point becomes the current point.

VI_RLine Draws a line from the current point to the current point displaced by the specified values, according to the current values of the merge, width, and dash pattern parameters. The current point is incremented by the displacement.

NOTE

VI_Line applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

clip(3G), dash(3G), merge(3G), width(3G)

This page intentionally left blank.

NAME

VI_Login, VI_Logout − begin logging subroutine calls and close a log file

SYNOPSIS

```
int VI_Login(filename)
        char *filename;           /* file to log to */

int VI_Logout()
```

DESCRIPTION

These subroutines begin logging subroutine calls and close the log file.

VI_Login   Specifies that subsequent subroutine calls are to be echoed into the specified file. If a log file is already open, *VI_Login* closes it before opening the new file; *VI_Login* overwrites an existing file. All orders to the display are logged until a logout call *(VI_Logout)* is issued. The log file may later be executed from within a program using *VI_Run* or on its own using *aedrunner*(1). It may also be examined with *aedjournal*(1).

VI_Logout   Closes the log file.

NOTE

VI_Log applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

aedjournal(1), aedrunner(1), init(3G), run(3G)

"The C Subroutine Interface for the IBM Academic Information Systems Experimental Display" in Volume II.

DIAGNOSTICS

VI_Login returns a negative value if there is an error, and a nonnegative value if the call is successful.

VI_Logout returns one of three values:

Value	Meaning
0	Normal completion
-1	Error in closing file
-2	No file found to close

This page intentionally left blank.

NAME

VI_Merge − set merge mode

SYNOPSIS

```
VI_Merge(merge)
    int merge;          /* merge mode */
```

DESCRIPTION

The merge mode is a number from 0 to 15 that specifies how the bits generated by primitives are to be combined with bits already on the screen, as shown in the following table:

Merge Mode	*Meaning*
0	OFF
1	NOR
2	NOT DATA AND SCREEN
3	NOT DATA
4	DATA AND NOT SCREEN
5	NOT SCREEN
6	XOR (NEQ)
7	NAND
8	AND
9	EQ
10	SCREEN (ignore)
11	NOT DATA OR SCREEN
12	DATA (replace)
13	DATA OR NOT SCREEN
14	OR
15	ON

The merge mode is initially set to 12, for replace mode. Data bits replace screen bits. The merge mode is simply an encoding of the logical function used to combine screen bits and data bits. Encoding the desired result of each of the combinations in the table below generates the merge mode that should be used to get that effect. For example, to *or* the data you are adding with the data already present on the screen, you would use a merge mode of 14:

Data Bit	1	1	0	0	
Screen Bit	1	0	1	0	
Example: OR mode	1	1	1	0	= 14

NOTE

VI_Merge applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

circle(3G), color(3G), line(3G), query(3G)

This page intentionally left blank.

NAME

VI_AMove, VI_RMove − move the current point

SYNOPSIS

```
VI_AMove(x,y)
        int x,y;            /* new point */

VI_RMove(dx,dy)
        int dx,dy;          /* displacement from old point */
```

DESCRIPTION

These functions move the current point; they do not change the screen. The current point is initially set to (0,0).

VI_AMove	Moves the current point to the specified coordinates.
VI_RMove	Moves the current point by the specified displacement.

NOTE

VI_Move applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

query(3G)

This page intentionally left blank.

NAME

VI_QClip, VI_QColor, VI_QDash, VI_QFont, VI_QMerge, VI_QPoint, VI_QWidth – query graphics parameters

SYNOPSIS

```
VI_QClip(lx,ly,hx,hy)
        short *lx,*ly;          /* top left corner of clipping area */
        short *hx,*hy;          /* bottom right corner */

VI_QColor(color)
        short *color;           /* current color, true for white */

VI_QDash(dash,dashlen)
        unsigned short *dash;           /* dash pattern */
        short *dashlen;         /* length of dash pattern */

VI_QFont(fontid,fontname)
        short *fontid;          /* current font ID */
        char *fontname; /* current font name */

VI_QMerge(merge)
        short *merge;           /* current merge mode */

VI_QPoint(x,y)
        short *x,*y             /* current point */

VI_QWidth(width)
        short *width;           /* line width */
```

DESCRIPTION

These subroutines return the current values of the graphics parameters. Each subroutine requires an address in which to store the value to be returned. All of these subroutines force transmission of graphics data in the current buffer.

VI_QClip	Returns the the current clipping rectangle.
VI_QColor	Returns the current color of the screen: 0 means that bits having the binary value "0" will be black on the screen; 1 means that bits having the binary value "1" will be black on the screen.
VI_QDash	Returns the current line dash pattern in the format described for *dash* (3G). If *dashlen* is 0, the lines are currently solid.
VI_QFont	Returns the ID and name of the current font. The font ID is 0 if no font has been set. The pointer *fontname* should point to a block of characters large enough to hold a file name along with a string-termination byte. If you know beforehand the size of your file name, you may allow only as many bytes as required. Be aware of the string-terminator byte; there must be room for it.
VI_QMerge	Returns the current merge mode in the format described for *merge*(3G).
VI_QPoint	Returns the location of the current point. This command is especially useful after *string*(3G) has been issued, since character definitions can change the current point in unpredictable ways.
VI_QWidth	Returns the current line width as a number between 1 and 16.

NOTE

VI_Query applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

clip(3G), color(3G), dash(3G), merge(3G), move(3G), string(3G), width(3G)

NAME

VI_MRead, VI_FRead – read display data

SYNOPSIS

```
VI_MRead(x,y,wd,ht,data)
        int x,y;                    /* top left corner of area */
        int wd,ht;                  /* dimensions of area */
        unsigned short *data;       /* first byte of data */

VI_FRead(x,y,wd,ht,filename)
        int x,y;                    /* top left corner of area */
        int wd,ht;                  /* dimensions of area */
        char *filename;         /* name of file to place image in */
```

DESCRIPTION

These functions read display data into memory or into a file. The area to be read must be completely on the screen. The current setting of the clipping window is ignored.

VI_MRead Reads the specified area of the screen into the array passed as *data*. Image bytes are in the same format as expected by *MImage*. If the screen color is white, the bits are inverted on readback to make the data read back independent of screen color. The area to be read must be completely on the screen.

VI_FRead Reads the specified area of the screen and places it in the specified file. The file has the same format as expected by *FImage*. If the window color is white, data bits are inverted to make the data independent of the screen color.

NOTE

VI_Read applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

image(3G)

This page intentionally left blank.

NAME

VI_Run − process a log file

SYNOPSIS

```
int VI_Run(filename)
    char *filename;          /* log file name */
```

DESCRIPTION

VI_Run executes the commands logged in the specified file; *filename* is the name of a log file that was created by *VI_Login*. Using *VI_Run* with a log file has the same effect as executing *aedrunner*(1) from within a program, allowing a series of orders which require much calculation to be figured only once, logged, then quickly retrieved when needed.

NOTE

VI_Run applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

aedjournal(1), aedrunner(1), log(3G)

DIAGNOSTICS

VI_Run returns 0 for normal completion, and -1 if it detects any kind of inconsistency or unexplained results in the file.

This page intentionally left blank.

NAME

VI_String – draw a string

SYNOPSIS

```
VI_String(s)
        char *s;            /* string to draw */
```

DESCRIPTION

VI_String draws the specified string at the current point. Since a character definition is really a sequence of other graphics commands (usually *VI_MImage* and *VI_RMove),* the way in which characters are positioned, stepped, and drawn depends on the font definition. Character definitions typically modify the current point.

NOTE

VI_String applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

font(3G)

"Defining Fonts" in "The C Subroutine Interface for the IBM Academic Information Systems Experimental Display"

This page intentionally left blank.

NAME

VI_Tile − tile a rectangle

SYNOPSIS

```
VI_Tile(wd,ht,twd,tht,tile)
        int wd,ht;              /* dimensions of rectangle */
        int twd,tht;            /* dimensions of tile */
        unsigned short *tile;   /* first byte of pattern */
```

DESCRIPTION

VI_Tile fills a rectangle of the specified dimensions with the specified pattern. The rectangle's top left corner will be at the current point. The tile pattern must follow the rules for images as explained in *image*(3G), and can be of any size. The tile pattern is aligned to multiples of *twd* and *tht,* not to the bounds of the tiled rectangle, so that rectangular subareas of larger figures can be tiled without regard to their bounds, and the tile patterns will match. The current point is unchanged.

A full rectangle black or white fill can be most quickly drawn by requesting a one-by-one tile. Clearly, only all ON or all OFF may be drawn with this method, but any merge mode may be used.

NOTE

VI_Tile applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

image(3G)

This page intentionally left blank.

NAME

VI_Width − set line width

SYNOPSIS

```
VI_Width(width)
     int width;           /* line width */
```

DESCRIPTION

VI_Width specifies a value between 1 and 16 that is to be the line width. Line width is initially set to 1.

NOTE

VI_Width applies only to the IBM Academic Information Systems experimental display.

SEE ALSO

line(3G), query(3G)

This page intentionally left blank.

Appendix C. High C ™ Programmer's Guide

ABSTRACT

This is a guide to the operation of the High C compiler as implemented for Academic Information Systems 4.2 for the IBM RT PC ("4.2/RT"). It contains:

1. INTRODUCTION

This is a guide to the operation of the High C compiler as implemented for Academic Information Systems 4.2 for the IBM RT PC ("4.2/RT").

The Compiler generates relocatable object modules directly, in contrast to most UNIX C compilers, which generate assembly files.

High C was designed to facilitate serious professional programming. It supports the draft ANSI Standard (ANSI document X3J11/85-102, August, 1985) and a few extensions.

C is a mixed-level systems language designed by Dennis Ritchie at AT&T's Bell Laboratories. It grew in popularity because of its use in implementing the UNIX operating system, its elegant (and deceptive) simplicity, and its close-to-the-machine features. As its popularity grew, many software developers have used it for real-world applications as well as systems software.

Later implementations of C were extended to add enumeration types and a few other features. More recently many extensions have been proposed to make C a safer language while still being consistent with the philosophy of the original language. Today there is a core language being standardized by the American National Standards Institute (ANSI).

High C includes what most likely will be ANSI Standard C and also provides extensions that were carefully designed to be consistent with the philosophy of C. Some of the best features of such other languages as Pascal, MetaWare's Professional Pascal, and Ada were incorporated as extensions. Incompatibilities were minimized by introducing a minimum of new key words and by retaining the original syntax. Yet the extensions are such that they will be flagged by any Standard-conforming compiler.

Portability. Standard C programs can be compiled with an ANSI option that turns off the extensions and reduces the language to the Standard core. Alternatively, such programs can be gradually upgraded by not choosing the ANSI option and using more and more extensions.

Safety, efficiency. While the close-to-the-machine features of C are available, High C supplies the new strong type-checking specified in ANSI C. In addition, the compiler provides many checking features usually available only in a separate "lint" program. Thus one gets both efficiency *and* reliability. It is an excellent language for both applications and systems programming.

Other important features and extensions include:

- three integer ranges and two floating-point precisions
- many compiler controls and options, including one for strict ANSI Standard checking
- nested functions complete with up-level references, as in Pascal
- nested functions passable as parameters to other functions, as in Pascal
- intrinsic functions, such as `_abs`, `_min`, `_max`, `_fill_char`, for efficiency
- many optimizations, some of which are normally found only in mainframe compilers, including:

    common subexpression elimination
    retention and reuse of register contents
    dead-code elimination
    cross jumping (tail merging)
    jump-instruction size minimization
    constant folding
    numerous strength reductions
    automatic allocation of variables to registers

This guide contains all system-specific information necessary for using the compiler effectively. Readers new to the product should scan the Table of Contents for an overview of the guide. Briefly, the next few chapters describe how to compile, link, and run, and how to use the compiler controls and options. There follow chapters on machine specifics, such as storage mapping and run-time organization. Next are chapters on communicating with programs written in other languages, listings, cross-references, defaults and limits, and error messages. An extensive index provides for quick reference to all sections that discuss or significantly relate to each topic.

This guide does *not* explain the High C language or its extensions. They are treated in the MetaWare *High C Language Reference Manual.* Neither this guide nor the manual attempt to teach C programming; consult the manual for references to several C textbooks.

2. INVOKING THE COMPILER

The **hc** command invokes the High C compiler, which translates C programs into executable load modules or into relocatable binary object modules suitable for linking with **ld**. The syntax of the command is:

```
hc [options]... files...
```

Any number of options and one or more files may be specified. Each option specified in the command applies to all the specified files for which it makes sense, except as noted below.

Several types of arguments are allowed. An argument ending with "`.c`" is taken to be a C source module. It is compiled and an object module is produced with the same name as the source except with "`.o`" substituted for "`.c`". The "`.o`" file is normally deleted after linking when a single-module C program is compiled and linked.

An argument whose name ends with "`.s`" is taken to be an assembly source module and is assembled, producing a "`.o`" file. Any other file specification is assumed to be an object module or archive library to be linked via **ld**.

All "`.o`" files are placed in the current working directory.

In general, **ld** is invoked if no compilation errors were detected and the **-c** option was not specified. The resultant load module is named "`a.out`" unless specified otherwise with the **-o** option (described below). Any argument beginning with a dash ("-") is taken as an option specification.

Example. The following command compiles the program in file `sort.c`, links it, and generates a load module named `sort`.

```
hc -o sort sort.c
```

2.1. Invoking the C Macro Preprocessor

The High C compiler has an integrated "inboard" macro preprocessor, documented in the *High C Language Reference Manual.* The preprocessor conforms to the proposed ANSI C Standard. However the "outboard" C macro preprocessor on most UNIX operating systems does not conform to the proposed Standard in some ways.

Since many C programs written for UNIX operating systems depend on minor idiosyncrasies of the outboard C preprocessor, the **-Hcpp/-Hnocpp** options are provided. The **-Hcpp** option causes the outboard preprocessor to be invoked on the source file sending the output to a temporary file, which then serves as input to the compiler. **-Hnocpp** suppresses this action. The compiler is provided with the **-Hcpp** option on by default.

2.2. Command Options

Below is a description of each compiler option. Any option that is not recognized by **hc** is assumed to be a linker option and is passed on to **ld**. Options applicable only to High C are prefixed with an **H**.

-Hansi Causes the compiler to accept only those programs conforming to the proposed ANSI Standard C language.
Note: Since the proposed ANSI Standard is under revision at the time of this writing, this option's primary function is to turn off the High C language extensions.

-Hasm Directs the compiler to produce an assembly listing of the generated code on standard output, by initializing the `Asm` toggle to `On`. The assembly listing is annotated with lines from the main source file but not with lines from any included files, for technical reasons. These lines appear as comments immediately preceding the corresponding assembly instructions. If the **-S** option (described below) is also specified, the generated

`.s` file is annotated with lines from the source file, and no listing is written on standard output.

-B*string* Finds substitute compiler executables in the files named *string* with the suffixes "`cpp`" and "`hccom`". If no *string* is given, the default `/usr/c/o` is used, i.e. the default is `/usr/c/ocpp` and `/usr/c/ohccom`.

-c Suppresses the invocation of **ld**, and forces an object file to be produced even if only one module is compiled.

-Hcpp Specifies that the outboard C macro preprocessor (`/lib/cpp`) is to be used, rather than the inboard preprocessor. **-Hcpp** is the default.

-Hnocpp Suppresses the use of the outboard C macro preprocessor in favor of the inboard preprocessor. See §2.1 for details.

-Hdebug Directs the compiler to emit additional symbol table information for the debugger **dbx**. This option is synonymous with **-g**.

-D*name*
-D*name=def*
Defines the name *name* to the preprocessor as if by "**#define**". If no *def* is given, the name is defined to be `1` (one). **Note:** There is no space between **-D** and *name*.

-E Specifies that the outboard C macro preprocessor is to be invoked and no compilation done. The preprocessor output is sent to standard output. (**-E** renders **-Hnocpp** irrelevant.)

-f Specifies that single-precision arithmetic is to be used in computations involving only **float** numbers. That is, floating-point operations are not to be performed in double precision, which is the default. Note that **float** arguments to non-prototyped functions are still converted to **double**, and function results declared to return **float** are returned as **double**. Some programs run much faster with this option, but beware of loss of significance due to lower-precision intermediate computations. (Also see toggle `Double_math_only` in §4 *COMPILER TOGGLES*.)

-g Directs the compiler to emit additional symbol table information for the debugger **dbx**. This option is synonymous with **-Hdebug**.

-I*dir* Specifies an alternate directory to be searched to locate an include file. This option may be specified several times to indicate several directories to be searched. If a particular file is not located after searching the specified directories, one or more standard directories are searched. See §3 *COMPILER PRAGMAS*. **Note:** There is no space between **-I** and the directory name *dir*.

-Hlines=*n* Causes a page eject to occur after every *n* lines written to standard output. The default of `60` is appropriate for most 6-lines-per-inch printers, which have a total of `66` lines per page. The setting of lines is intended to allow some blank space at page boundaries. When using 8-lines-per-inch, typically there are `88` lines per page, so **-Hlines** should be set to `80` or `82`. This option is used in conjunction with the **-Hlist** and **-Hasm** options. If *n* is `0`, no page ejects are emitted.

-Hlist Causes the compiler to generate a source listing on standard output. It works by initializing the `List` toggle to `On`. See §4 *COMPILER TOGGLES*.

-M Specifies that the outboard C macro preprocessor is to be invoked and `Makefile` dependencies are to be generated. The output is sent to standard output. No compilation occurs.

-m*x* Specifies a machine-dependent option. Currently available options are:

> **-ma** Specifies that the C library function `alloca` may be called from within the source file(s). `alloca` must extend the stack frame of `alloca`'s caller and needs certain information about the size of the caller's stack frame. This option makes the information available in the caller's data area. If `alloca` is called from a function that was not compiled with the **-ma** option, an error diagnostic is generated at run-time.

-o *output* Is passed on to the **ld** command and names the final executable output file *output*. When this option is used, any existing `a.out` file is left undisturbed. **Note**: White space *is* required after the **-o**.

-O Specifies that all optimizations supported by the compiler are to be performed on the generated code. This is the default unless **-g** is specified. Therefore, this option has meaning only when used in conjunction with **-g**.

-Hon=*toggle*
-Hoff=*toggle*
Turns a toggle `On` or `Off`. See §4 *COMPILER TOGGLES*.

-p Produces code that counts the number of times each function is called during execution. If **ld** is invoked, the profiling library `/usr/lib/libc_p.a` is searched in lieu of the standard C library `/lib/libc.a`. Also replaces the standard start-up function with one that automatically calls `monitor(3)` at the start and writes out a `mon.out` file. An execution profile can then be generated by use of `prof(1)`.

-pg Invokes a run-time recording mechanism as does **-p**, but keeps more extensive statistics and produces a `gmon.out` file. An execution profile can then be generated by use of `gprof(1)`.

-Hppo
-Hppo=*filename*
Specifies that the compiler is to invoke its inboard preprocessor only and send the results to "*filename*". If **-Hppo** alone is given, the preprocessor output is printed to the standard output. No object module is generated, nor is **ld** invoked. "**ppo**" can be read "pre-process only" or "print preprocessor output". The preprocessor output is suitable for input to the compiler.

With **-Hppo**, any `Include` pragmas are *not* processed, since **-Hppo** turns off all processing past the preprocessor, and a later phase of the compiler handles the `Include` pragma.

-R Makes all initialized static variables shared and read-only. This option is implemented by the assembler and therefore requires the compiler to emit an assembly source file. As a consequence, the **-Hasm** option is ignored.

-S Produces an assembly source file instead of an object file (for each source file). The assembly source is written into a file with the same name as the C source with "`.c`" replaced by "`.s`". The file is always placed in the current working directory. No object file is written, nor is **ld** invoked. **Note**: Unlike other compilers for UNIX operating systems, the High C compiler normally generates an object module directly, *without* producing an assembly file. The **-S** option essentially directs the last phase of the compiler to produce assembly source as the object code is generated. If the **-Hasm** option is also specified, the "`.s`" file is annotated with interlisted source file lines.

-U*name* Removes any initial definition of macro *name*. See **-D** above.

-v Causes the name of each subprocess to be printed as it begins to execute. (To get announcements of compiler-phase execution also, set **-Hoff**=`Quiet`.)

`-w`	Causes all warning messages from the compiler to be suppressed.
`-H+w`	Issues *ALL* warnings. The default is to issue only warnings that `pcc` would issue. This option comes highly recommended.

3. COMPILER PRAGMAS

The High C compiler provides "pragmas" (the term comes from Ada) that direct compiler operations. Pragmas control the inclusion and listing of source text, the production of object code files, the generation of optional additional program and debugging information, and so on.

3.1. Syntax of Pragmas

Compiler pragmas take one of the following general forms:

```
   pragma <Pragma_name>;  /* or */
   pragma <Pragma_name>(<Pragma_parameters>);
```

where `<Pragma_parameters>` is a list of constant expressions separated by commas. The number and types of the expressions are dependent upon the particular `<Pragma_name>`. A pragma can appear anywhere a statement or declaration can appear. See the *High C Language Reference Manual* for a specification of the precise placement of pragmas.

`<Pragma_name>`s are case insensitive.

3.2. Compiler Pragma Summaries

The following pragmas are available:

Pragma	Purpose
	Toggles* — see §4 *COMPILER TOGGLES:
`On,Off,Pop`	Turns `On` or `Off`, or reinstates a prior status of a compiler switch or "toggle".
	Externals* — see §7 *EXTERNALS:
`Alias`	Specifies the external name to be associated with a global identifier.
`Data`	Specifies the use of named blocks for data storage allocation. This is primarily intended for communicating with other languages.
	***Including Source Files* — see §3.3 below:**
`Include`	Includes the source of another file in the compilation unit.
`C_include`	Conditional form of `Include`.
`R_include`	Includes the source of another file in the compilation unit treating the path name as Relative to the directory of the file containing the pragma. This pragma has the same inclusion effect as the **`#include`** preprocessor directive.
`RC_include`	Conditional form of `R_include`.

3.3. `Include` Pragmas: Including Source Files

The **`#include`** preprocessor directive normally includes source text from an alternate file. The High C compiler supports pragmas with alternate search strategies for including files. This section describes the various strategies used to search for include files.

`Include` pragmas are processed by the compiler itself, not the macro preprocessor. Thus, we recommend using the **`#include`** directive rather than the `Include` pragma.

The `Include` pragma is used to include source from other files while the compilation unit is being compiled. The pragma operates slightly differently from the standard C **`#include`** directive. There are four forms of the `Include` pragma:

```
pragma    Include(<File_name>);
pragma  C_include(<File_name>);
pragma  R_include(<File_name>);
pragma RC_include(<File_name>);
```

where `<File_name>` is a string *constant* denoting the name of a file. ***Examples:***

```
pragma   Include("a_lot");
pragma R_include("dclns");
pragma C_include("math.h");
```

The `Include` pragma directs the compiler to include a file unconditionally. The `C_include` pragma causes the file to be included only if it has not been included before — "conditionally included". `R_include` has exactly the same effect as the standard C **#include** directive, i.e. it is a "relative include" (defined below). `RC_include` does a "conditional relative include".

The term *relative include* refers to an include in which the file is first sought in the directory of the file where the include pragma appears. If the file is not found in that directory, then any directories specified in any **-I** command line options are searched in order of appearance. See §2 *INVOKING THE COMPILER* for a description of the **-I** option. If the file is still not found, then one or more standard directories are searched.

A *non-relative include* refers to an include in which the current working directory is searched first irrespective of the location of the file in which the `Include` pragma appears. If the file is not found in that directory, then any directories specified in any **-I** command line options are searched in order of appearance. See §2 *INVOKING THE COMPILER* for a description of the **-I** option. If the file is still not found, then one or more standard directories are searched.

A file name specification that begins with "/" is assumed to be an absolute file reference and no directories are searched.

Preprocessor directive "**#include** "*filename*"" specifies a relative include.

Directive "**#include** <*filename*>" specifies that only the **-I** and standard directories are searched.

Warning. There should be nothing to the right of an `Include` pragma. After the `Include`d file is processed, processing resumes on the line immediately following the one containing the `Include` pragma. In effect the rest of the line is a comment.

Identity of file names. For the `C_include` and `RC_include` pragmas, file names, including path, are considered the same only if they are textually identical. Thus, the following two pragmas may cause two includes to occur:

```
pragma C_include(            "strings.h");
pragma C_include("/usr/include/strings.h");
```

even though both includes may refer to the same file.

Methodology. The primary use for conditional includes is to support modularity.

Assume file "`trees.h`" is merely a collection of declarations defining the interface to a `trees` module. Suppose further that `trees.h` makes reference to a type `Symbol` in another module defined in "`symbols.h`". If a standard "**#include** "`symbols.h`"" were placed within `trees.h`, a duplicate declaration of `Symbol` would occur in any compilation unit that `Include`d both `trees.h` and `symbols.h`. If, instead, a conditional `Include` were used in both `trees.h` and in any compilation unit including `symbols.h`, at most one copy of `symbols.h` would be included.

With conditional includes, each interface file `F` can conditionally include all other interface files that are necessary for the definition of the resources in `F`. Therefore any user of `F` can simply `Include F` and will automatically get other resources that are needed, without duplication.

4. COMPILER TOGGLES

Pragmas can be used to turn `On` and `Off` various compiler switches or "toggles". In such cases, the pragma syntax is simply:

```
pragma <Pragma_name>(<Pragma_parameter>);
```

The `<Pragma_name>` is either `On`, `Off`, or `Pop`, and the single `<Pragma_parameter>` is the name of the toggle to be affected. All compiler toggles are described in sections below.

`On` turns the toggle on; `Off` turns it off; and `Pop` reinstates it to a prior value. Toggles operate in a stack-like fashion, where each `On` or `Off` is a "push" of **on** or **off**, and a `Pop` "pops" the stack. The stack for each toggle is at least 16 elements deep, but no diagnostic is given if the stack overflows or underflows. ***Examples:***

```
pragma On (List);   -- Turns on the source listing.
pragma Off(Check_stack);  -- Turns off the run-time
pragma Off(List);         -- stack overflow checks.
pragma On (List);   -- Turns on the source listing.
pragma Pop(List);   -- Back to off for the listing.
pragma Pop(List);   -- Back to on  for the listing.
```

Recall that toggles can also be initialized on the command line, with **-Hon** and **-Hoff**. See §2 *INVOKING THE COMPILER.*

The default values, names, and meanings of the compiler toggles are described below.

Asm -- Default: Off

When `On`, causes a (pseudo-) assembly listing to be generated, annotated with source code as assembly comments. If the `Asm` toggle is to be turned `On` and `Off` over sections of a module, the pragma should appear among executable statements rather than declarations for best results; otherwise, the point at which the pragma takes effect may not be obvious.

Auto_reg_alloc -- Default: On

When `On`, causes the compiler to automatically allocate **auto** variables to registers. The compiler weights variables used within loops more heavily than those not so used in making its decision which variables to allocate to registers; furthermore it will not allocate to registers variables that are used too infrequently. See §5 *STORAGE MAPPING.*

Char_default_unsigned -- Default: On

When `On`, causes type **char** to be unsigned by default.

The ANSI Standard allows the type **char** by itself, i.e. without the adjectives **unsigned** or **signed**, to be either signed or unsigned. Of course, the types **unsigned char** and **signed char** can be used to explicitly control signedness.

Double_math_only -- Default: On unless the -f option is specified.

When `On`, causes floating-point operations to be performed in **double** precision.

When two operands of certain arithmetic operations are both of type **float**, the ANSI Standard permits an implementation to do one of two things: perform the operation using **float** arithmetic, in which case the result of the operation is of type **float**, or convert both operands to type **double** and use **double** arithmetic, in which case the result of the operation is of type **double**. When toggle `Double_math_only` is turned `Off`, the first option is used. When it is turned `On`, the second is used instead.

`Downshift_file_names -- Default: Off`

When `On`, causes the file name specification of any subsequent `Include` pragma to be interpreted as if it were in all lower case. This toggle is useful when moving source code to a UNIX operating system from others in which file name casing is not significant.

`Emit_line_table -- Default: Off`

When `On`, causes the compiler to add entries to the symbol table that associate source line numbers with object code addresses. Debuggers use this information to associate object code with source lines.

The **`-g`** (**`-Hdebug`**) command line option turns this toggle `On`.

Note: This toggle does not affect the size of the generated code, but it does add about eight bytes per statement to the object module's name list.

`Int_function_warnings -- Default: Off`

When `Off`, suppresses the warning message normally generated when a function returning **`int`** has no "**`return`** `exprn;`" statement within it, or a function returning **`int`** contains a "**`return;`**" within it.

This is to remove frequent warnings for old C source that did not use the reserved word **`void`** to indicate a function returning no result, since such functions return **`int`** by default.

`List -- Default: Off`

When `On`, causes the compiler to produce a listing on standard output. It is typically given when starting the compilation but may appear in the source file to turn the listing `On` or `Off` around a particular section of source.

`Literals_in_code -- Default: On`

When `On`, causes lengthy literals in a program to be placed in the code space rather than in the data space.

Note: Not all C literals can be placed in code. A string literal is a writable data item and hence cannot be placed in code; for such a literal `Literals_in_code` has no effect. See `Read_only_strings` below.

`Make_externs_global -- Default: On`

When `On`, any local declaration of an object with storage class **`extern`** is made global if there is not already a global declaration of the object. Early C compilers improperly promoted an **`extern`** declaration within a function to the global scope. This toggle supports programs depending upon that "feature".

`Optimize_for_space -- Default: Off`

When `On`, causes the generation of more space-efficient but potentially less time-efficient code.

`Optimize_xjmp -- Default:On`

When `On`, enables the cross-jumping optimization. While an effective space-saving optimization that leaves execution time invariant, it slows the code generator a little and can produce code that is difficult to debug. See Appendix A *CROSS-JUMPING OPTIMIZATIONS* of this guide for more information on the specifics of this optimization. See also the `Optimize_xjmp_space` toggle below.

`Optimize_xjmp_space -- Default: On`

When `On`, enables a cross-jumping optimization that saves space but always at the expense of time. This toggle takes effect only if `Optimize_xjmp` is also `On`. This optimization slows the code generator a little and can produce code that is difficult to debug. See Appendix A *CROSS-*

JUMPING OPTIMIZATIONS of this guide for more information on the specifics of this optimization. See also the `Optimize_xjmp` toggle above.

Parm_warnings -- Default: On

When `On`, causes the compiler to produce warnings whenever arguments are passed to a non-prototype (old-style) function `F` do not match in type with the declared formal parameters of `F`. Frequently this inconsistency is a source of disastrous or difficult-to-find bugs. ***Example:***

```
double square(x) double x; {return x*x;}
(...)
printf("%d\n",square(3));
```

The call to `square` passes the integer `3`, not the double `3.0`, and the compiler issues a warning. The C language definition *prohibits* the compiler from casting `3` to a **double** before passing it.

To eliminate the compiler warnings, turn `Off` the toggle `Parm_warnings`. We recommend, however, that the program text be repaired to eliminate the offending function calls rather than eliminating the potentially useful feedback from the compiler.

PCC_msgs -- Default: On

When `On`, the diagnostic capabilities of the compiler are reduced to the **pcc** level in that the following warnings are not emitted:

```
Function called but not defined.
Function return value never specified within function.
This "return" should return a value of type ttt
   since the enclosing function returns this type.
"=" used where "==" may have been intended.
Only fields of type "unsigned int" or
   "unsigned long int" are supported.
External function is never referenced.
Declared type is never referenced.
```

The next four messages are suppressed for global variables when `PCC_msgs` is `On`:

```
Variable is never used.
Variable is referenced before it is set.
Variable is referenced but is never set.
Variable is set but is never referenced.
```

When all warnings are enabled in High C, code must be "squeaky clean" to get through the compiler without a warning. Some users have code that was designed with a compiler that is not so demanding, and would prefer fewer prods from the compiler. Hence the `PCC_msgs` toggle is supplied.

Pointers_compatible -- Default: Off

When `On`, allows pointers of any type to be compatible with each other. Although this is in violation of the ANSI Standard and High C specifications, many old C programs improperly assign pointers of different types to each other. This toggle allows such programs to be compiled without modification.

Pointers_compatible_with_ints -- Default: Off

When `On`, allows pointers of any type to be compatible with **ints**. Although this is in violation of the ANSI Standard and High C specifications, many old C programs improperly assign pointers and **ints** back and forth. This toggle allows such programs to be compiled without modification.

ANSI and High C disallow this dangerous practice because pointers are not necessarily the same size as **ints** on all machines. The programmer should ensure that intermixed pointer and **int** values have the same size; otherwise a pointer stored in an **int** may not be retrieved as expected later.

`Print_ppo -- Default: Off`

When `On`, causes preprocessed input to be written to standard output. With this toggle, it is possible to print what the compiler proper receives over a local area of source code. A use would be to turn the toggle `On` prior to a complex macro invocation and `Off` after it, to verify that the macro expands as expected. **Note:** This toggle is ignored unless **`-Hnocpp`** is specified or is the default.

`Print_protos -- Default: Off`

When `On`, causes the compiler to write to standard output a new, prototype-style function header for each function definition. This toggle aids in the conversion of C programs to use the new ANSI prototype syntax derived from the C++ language. For example, for the function definition

```
int f(x,y,z) int *x,z[]; double (*y)(); {...}
```

the compiler produces

```
int f(int *x, double (*y)(), int *z);
```

The old function header can then be replaced with the generated one.

There are some minor pitfalls in having the compiler automatically generate prototype headers. One is illustrated above: array parameters, according to the semantics of C, are converted to pointer parameters. Second, **`enum`** types are converted to their representation type (one of the **`int`** types). Finally, the compiler does not distinguish the type specifier **`char`** from the **`signed`**- or **`unsigned-char`** that **`char`** alone stands for. This means that for High C, both **`char`** and **`unsigned char`** are printed as **`unsigned char`**. The **`enum`** and **`char`** problems can be avoided by using **`typedef`**s, and using a **`typedef`** name for the parameter's type.

`Print_reg_vars -- Default: Off`

When `On`, causes the compiler to write to standard output the mapping of variables to registers. This saves the programmer the trouble of looking at the generated code to discover such information.

`Public_var_warnings -- Default: On`

When `Off`, suppresses the warning messages:

```
Variable is never used.
Variable is referenced before it is set.
Variable is referenced but is never set.
Variable is set but is never referenced.
```

for all variables exported, i.e. non-automatic variables not declared **`static`** or **`extern`**.

These warning messages occur only for such variables that are not declared within a **`#include`**-d file. If one adheres to the discipline that all imported variables are defined in included files, the message will not occur.

`Quiet -- Default: On`

When `Off`, causes each compilation phase to be announced in turn as the compilation progresses. (This toggle is not turned `Off` by **`-v`**.)

`Read_only_strings -- Default: Off`

When `On`, string literals are considered true literals. Identical string literals appear in the object code only once and the `Literals_in_code` toggle (see above) takes effect for string literals, causing them to be placed in code.

C string literals are not true literals since they are writable data items. This means that they cannot normally be placed in code space. Furthermore, two identical C string literals must normally be duplicated in a program's object code, since one might be modified and the other not.

To avoid this, use `Read_only_strings` and `Literals_in_code`. These two toggles cause C string literals to be placed in code.

The **-R** option turns `Read_only_strings` `On` initially.

`Summarize -- Default: Off`

When `On`, causes the production of summaries of compilation activities. The summaries are produced at various stages of compilation.

`Warn -- Default: On`

When `Off`, causes warning messages to be suppressed. The **-w** option turns `Warn Off` initially.

5. STORAGE MAPPING

5.1. Data Types in Storage

The table below summarizes the size and alignment of various C data types, and whether a variable of the type can be allocated to a register.

Data Type	Size	Alignment	Allocable
char	1 (bytes)	1 (bytes)	Y
short int	2	2	Y
int	4	4	Y
long int	4	4	Y
float	4	4	Y
double	8	4	Y
long double	8	4	Y
enum	See Note 3.	See Note 3.	—
Pointer	4	4	Y
Full-function[1]	8	4	N
T[n]	n***sizeof**(T)	Same as T	N
struct{...}	See Note 1.	See Note 2.	N
union {...}	See Note 1.	See Note 2.	N

Note 1: The size of a **struct** is the sum of the sizes of its fields, including alignment padding between fields. It is padded so that its size is evenly divisible by its alignment. The size of a **union** is the size of the biggest field, padded so that its size is evenly divisible by its alignment.

Note 2: A **struct** or **union** is aligned according to the requirements of the most stringent member.

Note 3: An **enum** type is the same as **char, short, int,** or **long int.** The smallest of these types is chosen such that all values of the **enum** type can be represented.

Bit fields. Only unsigned bit fields are supported. A bit field may not exceed 32 bits and is packed in each consecutive byte from left to right. A bit field must fit within a four-byte word that is aligned to a four-byte boundary. Padding is added where appropriate to make this true.

A bit field of length zero causes alignment to occur at the next full-word boundary, i.e. where an **int** would be aligned.

A bit field that is byte-aligned and one byte long is treated as if it were type **unsigned char**. One that is two-byte-aligned and one-to-two bytes long is treated as if it were an **unsigned short.** Finally, a three-to-four byte field is treated as an **int.** These treatments afford efficient access.

5.2. Storage Classes

Each **static** variable is placed in either the BSS section or the DATA section — the latter if it is initialized.

Each global variable with no **extern** specifier that is not initialized is defined as a common block; if it is initialized, it is mapped into the DATA section and given the *global* attribute. Each **extern** variable is given the *global* and *undefined* attributes.

Each **auto** variable is assigned either to a machine register or to storage in the routine's "stack frame". See §6 *RUN-TIME ORGANIZATION.* The compiler chooses which of the **auto**-classed

----

1. A full-function value is a High C extension. It consists of a function address and a static link. See the *High C Language Reference Manual* for details.

variables to place in registers based upon the variable's type, frequency of reference, and whether or not the "`&`" operator is ever applied to it.

Each **`register`** variable is treated as an **`auto`** variable except that it is given extra weight in assignment to a machine register.

Be warned that use of library functions `setjmp` and `longjmp` can produce unpredictable results in the context of register variables.

6. RUN-TIME ORGANIZATION

High C adheres to the standard linkage convention established by 4.2/RT[2]. This chapter presumes knowledge of the RT architecture and assembly language. Throughout this chapter the term "word" denotes a four-byte storage unit.

6.1. Register Usage

Certain registers, such as `r1`, have specific uses throughout execution; others, such as `r15`, are used during a function call and are free at other times. The following table defines register usage at the call interface.

Reg-ister	Saved over call	Use
`r0`	no	called-function data area pointer
`r1`	yes	stack pointer (caller's frame pointer)
`r2`	no	argument word 1 and returned value
`r3`	no	argument word 2 and lower half of a returned double value
`r4`	no	argument word 3
`r5`	no	argument word 4
`r6-r12`	yes	register variables, etc.
`r13`	yes	frame pointer
`r14`	yes	data area pointer
`r15`	no	return address
`mq`	no	multiply/divide register

In addition, floating-point registers 0, 1, and 6 are not saved over a call; registers 2-5 are preserved.

6.2. The Data Area

Each C function has an "entry point" and a "data area". Both must be referenced at the point of a call.

The *entry point* is where the code of the function begins. The *data area* (also called a "constant pool", which is a misnomer,) contains strings, function addresses, and other literals.

A function `foo` normally has an entry point named `_.foo` and a data area named `_foo`.

The call instruction sequence sets `r0` to the address of the called-function's data area. The first word in the data area is the entry point of the called function. The word following supports the code profiling option (**-p**), and if present must be initialized to zero; the third word, also optional, supports "`alloca`" storage allocation.

In the function prologue code, `r0` is copied to `r14`; thus, `r14` is used to address the associated data area from within a function.

The data area is placed in the `DATA` section so that `r14` may be used as a base for referencing local static variables. Static variables are usually mapped before the various data areas; therefore, static variable references employ negative offsets from `r14`.

When a pointer to a function is assigned the "value" of a function, it is actually assigned the address of the function's data area. The first word of the data area always contains the entry point, i.e. the address of the first instruction of the function.

----

2. Portions of this chapter copyright International Business Machines Corporation, 1986. Excerpts by permission, from the manual *Academic Information Systems 4.2 for the IBM RT*. More information may be found in "4.2/RT Linkage Convention" in Part II, Supplementary Documents.

6.3. Stack Frame Layout

The stack holds frames for currently active functions. It is word-aligned and grows downward. `r1`, the "stack pointer", indicates the low address of the stack frame of the currently executing function.

A stack frame is divided into the following areas, highest address first:

a) space for incoming argument list (4 words)
b) linkage area (4 words; reserved)
c) static link (1 word)
d) general register save area (16 words maximum)
e) floating-point register save area (8 words maximum)
f) local variables and temporary storage
g) words 5 through n of out-going argument lists

The *static link* applies to a function that is nested within another function; it is the address of the enclosing function's stack frame. (Nested function definitions, as in Pascal, are a High C extension to Standard C.) The static link is used to do "up-level addressing", i.e. referencing local variables of containing functions. While executing level-one functions, the static link field is uninitialized.

The caller's return address (`r15`) is saved at a fixed offset of 10 words below the top of the stack frame, at the top of the general register save area.

The floating-point register save area holds up to 4 double-word registers ending with register 5. It is empty if no such registers need preserving.

The compiler uses `r13` to reference the top of the stack frame. Since it is more efficient to access variables with small positive displacements, the compiler often biases the value of `r13` to improve the code for local variable accesses — see §6.7 *Prologue* below for more information.

6.4. Argument Passing

Arguments are word-aligned and allocated to consecutive words on the stack. The list lies across frame boundaries: words 1-4 are allocated in the top of the callee's frame, and the remainder are in the bottom of the caller's frame, which is adjacent. In a call, words 1-4 are actually passed in registers `r2`-`r5`.

Arguments are passed as follows, based on argument type:

- An **`int`** is passed in a single word.
- A **`long`**, **`short`**, pointer, or **`char`** is treated as an **`int`** and passed in a word.
- A **`double`** is passed in two consecutive words.
- A **`float`** is converted to **`double`** and passed in two consecutive words, unless it is being passed to a prototyped function that was declared to receive a **`float`**, in which case it is passed in a word.
- A structure is aligned to a word and left justified, except for a structure of 1, 2, or 3 bytes, which is right justified.
- A pointer to a function is passed as a pointer to the function's data area.
- A full-function value[3] is passed as two words. The first contains the address of the data area; the second contains the static link.

If a function is declared as returning a structure, the caller passes the address of a result area in `r2`. The first word of the explicit argument list is passed in `r3`. Subsequent arguments are shifted accordingly.

----

3. A full-function value is a High C extension. It consists of a function address and a static link. See the *High C Language Reference Manual* for details.

6.5. Function Results

A result is returned from a function in one of three ways, depending on the function's return type:

- An **int**, **long**, **short**, pointer or **char** is returned in r2.
- A **double** is returned in r2 and r3.
- A **float** is widened and returned as a **double**.
- A structure result or full-function value is returned by moving it into the area pointed to by the first word in the argument list (in r2 on entry).

6.6. Calling Sequences

A call of a known function foo first prepares the argument list, then executes the following:

```
balix r15,_.foo          # Call.
l     r0,$.long(_foo)    # Get its data area
                         # pointer, r14 relative.
```

If the function being called is nested within another function (High C, not plain C), the caller stores the static link, i.e. the frame pointer of the enclosing function, into -36(r1) before executing the **balix**.

Note that the address of the data area of the function being called is in the data area of the caller and is referenced off of r14.

A call to a function via a function pointer is done as follows. Recall that a function pointer addresses the function's data area. If the pointer is in r8, typical code is:

```
ls    rt,0(r8)          # Get address of entry point.
balrx r15,rt            # Call.
mr    r0,r8             # Load r0 with data area address.
```

6.7. Prologue

Prologue code saves the caller's registers, establishes the frame pointer (r13), and obtains stack space for the stack frame. Typical code is:

```
_.foo:  stm   rn,-76+(n-6)*4(r1)# Save caller's
                                #   registers.
        mr    r14,r0            # Set up addressability
                                #   to data area.
        mr    r13,r1            # Set up frame pointer.
        cal   r1,frame_size(r1) # Allocate stack frame.
```

Here *n* (6≤*n*≤13) is the register number of the first general register to be saved, and *frame_size* is the size of the stack frame (word-aligned) including the space required for the caller's save area. Other instruction sequences are needed for frame sizes larger than 32,767 bytes.

If floating-point registers need saving, the following code is inserted before the allocation of the stack frame:

```
cal   r12,-fsave(r1)    # Load r12 with address
                        #   of save area.
bali  r15,fpstmn        # Call routine to
                        #   save registers n-5.
```

where *fsave* is the offset of the floating-point register save area that resides immediately below the general register save area. *n* (2≤*n*≤5) is the first of the floating-point registers 2 through 5 needing to be saved.

Because the fpstm*n* functions modify general registers r10 and r11, the registers must be saved by the previous **stm** instruction.

As noted earlier, `r13` may be biased by some negative amount so as to improve code references to stack frame variables. For example, "**mr** `r13,r1`" may be replaced with "**cal** `r13,-80(r1)`".

6.8. Epilogue

The epilogue restores the caller's environment and returns control. Typical code is:

```
mr    r1,r13            # Restore stack pointer.
lm    rn,-76+(n-6)(r1)  # Restore general reg.
br    r15               # Return to caller.
```

where *n* is the same value as in the **stm** instruction of the corresponding prologue.

If floating-point registers are involved, these instructions appear before the **lm** instruction:

```
cal   r12,fsave(r1)     # Load r12 with address
                        #   if save area.
bali  r15,fplmn         # Restore floating
                        #   registers n thru 5.
```

6.9. Assembler Issues

Temporarily, all modules linked by **ld** must have the global symbol `.oVncs` defined as an absolute with value `0`. This distinguishes modules using an earlier linkage convention that is now obsolete. From assembly language, the symbol can be defined via:

```
.globl .oVncs
.set   .oVncs,0
```

7. EXTERNALS

The names of variables and functions that are communicated across module boundaries are normally made global in the resultant object module. In large programs there may be hundreds or even thousands of such names, so name conflicts are likely to occur.

Unfortunately neither C nor most linkers provide for a structured name space — for named packages of resources, for example. Thus the well-chosen "internal" names in a program may not also be usable as "external" names (those known to the linker) as they should be. Thus some method of aliasing internal names to externals is needed, and High C provides it.

It is important to be able to alias such names to avoid conflicts in the linker's external symbol dictionary, rather than being forced to pervert the internal names themselves. It is the internal names that are most important to be well-chosen "containers of meaning", for program maintainability.[4]

7.1. The `Alias` Pragma

This pragma specifies, for a specific internal name, another name for external or public purposes. It is the alternate name that appears in the object module. The form of the `Alias` pragma is as follows:

```
pragma Alias(<Internal_name>,<External_name>);
```

where `<Internal_name>` is the function or variable identifier being aliased and `<External_name>` is a *constant* string expression whose value denotes the alternate or external name.

The `Alias` pragma must appear *in the scope of* the declaration of the internal name. ***Example:***

```
void Initialize();

    pragma Alias(Initialize,"x_initialize");
    /* The function Initialize is referenced in the */
    /* object-module name list as "x_initialize".   */

int A;
    pragma Alias(A,"A");
    /* "A" is referenced in the name list as "A"    */
    /*  instead of "_A".                            */
```

7.2. Data Segmentation: the `Data` Pragma

Audience. This section may be skipped unless there is an interest in either (a) communicating with programs written in Professional Pascal or (b) using a data communication convention different from that of standard C.

Communication between separately-compiled modules is achieved by using the **extern** storage class in C. Multiple defining declarations of a variable `x` are allowed, as long as at most one of them initializes `x` (thus the **extern** storage class is not required).

The `Data` pragma provides an alternative method of sharing data, using named blocks. Its general usage is illustrated by:

```
pragma  Data(class,"blockname");
int         X,Y,Z: ...;
...  /* Other normal C declarations may appear here. */
pragma  Data;
        /* "Turns off" the prior Data pragma. */
```

----

4. The external names are also important in that respect, but we believe that the proper solution is a "module interconnection language" and associated linker with a structured dictionary to match the overall structure of the program.

where *class* is one of Common, Import, or Export, and "blockname" is a constant string expression. The ending Data pragma has no parameters.

Only the given block name is made known to the linker as a global symbol: each variable is addressed at a fixed offset within the block. When the Import class is specified, the symbol is given the *undefined global* attributes and a value of 0; when Export, the symbol is defined in the module's BSS or DATA segment and given the *global* attribute. When Common, the symbol is flagged as a named common block, i.e. given the *undefined global* attributes and a value that is equal to its length.

Scope. Each Data pragma must be terminated or "turned off" as illustrated above *in the same scope* in which it is turned on. The storage class specification applies only to variable declarations between the specification and its termination, *not to any variable declared within embedded function definitions* (a High C extension). That is, variables declared at lower levels — local to surrounded (nested) function declarations — are not affected: at a function declaration, any active Data pragma temporarily becomes inactive and the default applies through the end of the function.

A compile-time warning is issued if a Data pragma is specified when a prior Data pragma is still active (in which case the subsequent pragma applies), or if a Data pragma is active at the end of a function declaration or at the end of a compilation unit. Thus Data pragmas cannot be nested within a single function, though they can be nested if they apply to the local variables of distinct functions.

Example:

```
pragma Data(Common,"BLOCK");
int    Tables_are_loaded: Boolean;
struct {...} Tables;
pragma Data;
```

Here, the names, Tables and Tables_are_loaded, are mapped at consecutive displacements (subject to boundary alignment) within the common block "BLOCK".

8. ASSEMBLY LANGUAGE COMMUNICATION

8.1. Assembly Routines

The Sections *Prologue* and *Epilogue* in §6 *RUN-TIME ORGANIZATION* describe the code that an assembly routine must execute in order to be callable from C. In short, an assembly routine should be coded according to the following guidelines. Symbols in *italics* are to be filled in appropriately.

```
         .text
         .globl   _.name
         .globl   _name
_.name:  stm      rn,-76+(n-6)*4(r1)
         mr       r14,r0
         mr       r13,r1
         cal       r1,frame_size(r1)
#        The body of the routine goes here.
         mr       r1,r13
         lm       rn,-76+(n-6)*4(r1)
         lr       r15
         .align   2
_name:
         .long    _.name
```

where *name* is the function's name as referenced from C; *n* (6≤*n*≤13) is the register number of the first general register to be saved; *frame_size* is the size of the stack frame (word-aligned) including the space required for the caller's save area.

For a description of how arguments are passed and how function results are returned, see §6 *RUN-TIME ORGANIZATION.*

8.2. Function Naming Conventions

An identifier that is global, i.e. accessible across module boundaries, must have information provided to the linker that associates its name with its address. This is done by placing a corresponding name in the *name list* of the object module and giving it the "global" attribute.

There are two names associated with every function: one referencing the entry point and one referencing the associated data area. The name that references the data area of a C function `foo` is `_foo`; the entry point is referenced by `_.foo`.

8.3. Examples: Calling Assembly from C

Example #1:

High C:

```
extern void and(int *dest, int *src, int len);
void main()
   {
   int a[256],b[256];
   ...
   and(a,b,256);
   ...
   }
```

Assembly:

```
         .text
         .globl  _and
         .globl  _.and
         .align  2
_and:    .long   _.and
_.and:
         stm     r13,-48(r1)
         mr      r14,r0
         mr      r13,r1
         cal     r1,-48(r1)
L:       cis     r4,0
         jle     exit
         ls      r0,0(r2)
         ls      r5,0(r3)
         n       r0,r5
         sis     r4,1
         bx      L
         sts     r0,0(r2)
exit:    mr      r1,r13
         lm      r13,-48(r1)
         br      r15
```

Since the assembly routine does not modify non-volatile registers and has a zero-length stack frame (except for the caller's save area), it can be optimized to the following:

```
         .text
         .globl  _and
         .globl  _.and
         .align  2
_and:    .long   _.and
_.and:
L:       cis     r4,0
         bler    r15
         ls      r0,0(r2)
         ls      r5,0(r3)
         n       r0,r5
         sis     r4,1
         bx      L
         sts     r0,0(r2)
```

However, if an exception should occur in the optimized routine, e.g. an invalid address passed in, the debugger may be hampered in identifying the context.

Example #2:

High C:

```
extern char peek(char *adr);
void main(){
   char  b;
   ...
   b = peek(0x8000);
   ...
   }
```

Assembly:

```
        .text
        .globl  _peek
        .globl  _.peek
        .align  2
_peek:  .long   _.peek
_.peek: lc      r2,0(r2)        # Return the byte.
        br      r15
```

8.4. Example: Calling C from Assembly

To call a C function `foo` from assembly language, first store the arguments in `r2` through `r5` (and on the stack at `0` off of `r1` if applicable) and then execute the following two instructions.

```
        balix  r15,_.foo
        l      r0,x(r14)
```

where `x(r14)` references a memory location containing the address of `_foo`.

Example:

High C:

```
void write_string(char *s)
   {
   printf("%s\n",s);
   }
```

Assembly:

```
        .text
        .globl  _write_string
        .globl  _.write_string
_name:  .long   _.name
        .long   _write_string
_.name: stm     r13,-48(r1)
        mr      r13,r1
        mr      r14,r0  # Set up reference
        ...             #    to data area.
        get     r2,msg
        balix   r15,_.write_string
        l       r0,4(r14)
                        # i.e. _name + 4
        ...
msg:    .asciz  "This is a message."
```

8.5. Data Communication

A global variable "x" appears in the name list as "_x", unless specified otherwise with an `Alias` pragma — see §7 *EXTERNALS.*

§5 *STORAGE MAPPING* explains how the various C data types are mapped into storage. Note that uninitialized global variables without the **extern** qualifier are actually defined as individual common blocks. The following examples illustrate the sharing of variables across C and assembly modules:

High C:

```
int alpha,beta;
char hextable[] = "0123456789ABCDEF";
extern  char *names[]; /*A read-only table of names.*/
extern  short status;
```

Assembly:

```
        .comm   _alpha,4
        .comm   _beta,4
        .globl  _hextable   # Imported from C.
        .text
        .globl  _names      # Read-only;
_names: .long   L01         # in text segment.
        .long   L02
        .long   L03
        .long   0
L01:    .asciz  "alfred"
L02:    .asciz  "bonny"
L03:    .asciz  "charlie"

        .data
        .globl  _status
_status:
        .short  0
```

High C provides the ability to map more than one variable into a named block, e.g. a common block as in FORTRAN. This facility is provided by the `Data` pragma and is documented in §7 *EXTERNALS*. The following illustrates how such a common block may be accessed from assembly language.

C Common Block Definition:

```
pragma Data(Common,"BLOCK_NAME");
int    a,b;
char   c,d;
short  e;
pragma Data;
```

Assembly Language Equivalent:

```
        .comm   BLOCK_NAME,12
        .set    a,0
        .set    b,4
        .set    c,8
        .set    d,9
        .set    e,10
        ...
Usage:
        get     r2,BLOCK_NAME
        l       r3,a(r2)    # Load value of a.
        l       r4,b(r2)    # Load value of b.
        lc      r5,c(r2)    # Load value of c, etc.
```

Note that variables `a,b,c,d,` and `e` are *not* global; that is, they do not appear in the name list with the "global" attribute. The only name that appears in the name list is `BLOCK_NAME`.

9. LISTINGS

9.1. Pragmas `Page, Skip, Title`

To cause `n` page ejects at some point in the listing, insert:

```
pragma Page(n);  /* where n is the number of ejects. */
```

To cause `n` lines to be blank at some point in the listing, insert:

```
pragma Skip(n);  /* where n is the number of blanks. */
```

To cause a title `T` to appear at the top of each successive page, place the following pragma in the source:

```
pragma Title(T); /* where T is a string constant. */
```

Each successive `Title` pragma changes the title for the next pages; therefore the title does not appear on the first page.

9.2. Format of Listings

Ruler. The first line after any header and title lines on each page is a "ruler" that defines three fields for each line. The fields are for: (1) three level numbers, (2) the line number, and (3) the line contents. The ruler is as follows:

```
Levels LINE#|----+----1----+----2----+----3----+----4----+----5---
```

Level-numbers can be used to find a missing `}` or comment terminator when a message such as "`Unexpected end-of-file.`" is produced by the compiler. All three level-numbers are initially zero, but they are printed as blank rather than "`0`".

The first level-number indicates the scope nesting level for **`struct`** or **`union`** declarations.

The second level-number indicates the statement nesting level. It is incremented at the beginning of each `{` and is decremented at the corresponding `}`.

The third level-number indicates the structure initialization nesting level. It is incremented at the beginning of each `{` and decremented at the corresponding `}`.

Include files. A first-level `Include` file named `File_name` is indicated as starting after a line containing "`+(File_name`" in the line number field, and ending just before a matching "`+)File_name`" line. The included lines have "+" in the leftmost column of the line-number field, and those lines are numbered independently of the main source file.

An `Included` file inside an `Include` file has an extra "+" on each of its lines for each level of inclusion, except that line numbers take precedence over "+"s in the line-number field if and when the "+"s would otherwise intrude into the field.

The listing facility should be used in conjunction with the **`-Hnocpp`** option. Otherwise the output of the outboard C preprocessor will be listed; each `Include` file specified with the **`#include`** preprocessor statement is back substituted with no indication on the listing.

Example. Since a picture is worth a thousand words, a sample program listing appears on the next two pages, enhanced with boldface reserved words and followed by the optional assembly listing requested by **`-Hasm`** on the following compile command line:

```
hc queens.c  -Hlist  -Hasm  -Hnocpp
```

```
MetaWare High C Compiler 1.3   07-Jul-86 17:13:14       queens.c  Page 1
Copyright (C) 1983-86 MetaWare Incorporated.

Target: 4.2/RT   (Code generator 1.3)
Levels  LINE # |----+----1----+----2----+----3----+----4----+----5----+
             1 |/* From Wirth's Algorithms+Data Structures = Programs.*/
             2 |/* This program is suitable for a code-generation     */
             3 |/* benchmark, especially given common sub-expressions */
             4 |/* in array indexing.  See the Programmer's Guide for */
             5 |/* how to get a machine code interlisting.           */
             6 |
             7 |pragma  Title("Eight Queens problem.");
             8 |
             9 |typedef enum {False,True} Boolean;
            10 |typedef int  Integer;
            11 |
            12 |#define Asub(I) A[(I)-1] /* C's restriction that array*/
            13 |#define Bsub(I) B[(I)-2] /* indices start at zero     */
            14 |#define Csub(I) C[(I)+7] /* prompts definition of     */
            15 |#define Xsub(I) X[(I)-1] /* macros to do subscripting.*/
            16 |                         /* Pascal equivalents:       */
            17 |static Boolean A[ 8]; /* A:array[ 1.. 8] of Boolean */
            18 |static Boolean B[15]; /* B:array[ 2..16] of Boolean */
            19 |static Boolean C[15]; /* C:array[-7.. 7] of Boolean */
            20 |static Integer X[ 8]; /* X:array[ 1.. 8] of Integer */
            21 |
            22 |void Try(Integer I, Boolean *Q) {
  1         23 |   Integer J = 0;
  1         24 |   do {
  2         25 |      J++; *Q = False;
  2         26 |      if (Asub(J) && Bsub(I+J) && Csub(I-J)) {
  3         27 |         Xsub(I) = J;
  3         28 |         Asub(J) = False;
  3         29 |         Bsub(I+J) = False;
  3         30 |         Csub(I-J) = False;
  3         31 |         if (I < 8) {
  4         32 |            Try(I+1,Q);
  4         33 |            if (!*Q) {
  5         34 |               Asub(J) = True;
  5         35 |               Bsub(I+J) = True;
  5         36 |               Csub(I-J) = True;
  5         37 |               }
  4         38 |            }
  3         39 |         else *Q = True;
  3         40 |         }
  2         41 |      }
  1         42 |   while (!(*Q || J==8));
  1         43 |   }
            44 |pragma Page(1); /* Page eject requested. */
```

```
MetaWare High C Compiler 1.3  07-Jul-86 17:13:14       queens.c  Page 2

                Eight Queens problem.
Levels  LINE # |----+----1----+----2----+----3----+----4----+----5----+
            45 |void main () {
  1         46 |   Integer I; Boolean Q;
  1         47 |   printf("%s\n","go");
  1         48 |   for (I =  1; I <=  8; Asub(I++) = True);
  1         49 |   for (I =  2; I <= 16; Bsub(I++) = True);
  1         50 |   for (I = -7; I <=  7; Csub(I++) = True);
  1         51 |   Try(1,&Q);
  1         52 |pragma Skip(3); /* Skip 3 lines. */



  1         53 |   if (Q)
  1         54 |      for (I = 1; I <= 8;) {
  2         55 |         printf("%4d",Xsub(I++));
  2         56 |         }
  1         57 |   printf("\n");
  1         58 |   }
```

```
MetaWare High C Compiler 1.3  07-Jul-86 17:13:14  queens.c  Page 3

                Eight Queens problem.
Addr  Object    Source Program and Assembly Listing
                        .globl  .oVncs
                        .set    .oVncs,0
                        .globl  _printf
                        .globl  _.printf
        #/* From Wirth's Algorithms+Data Structures = Programs */
        #/* This program is suitable for a code-generation     */
        #/* benchmark, especially given common sub-expressions */
        #/* in array indexing.  See the Programmer's Guide for */
        #/* how to get a machine code interlisting.            */
        #pragma  Title("Eight Queens problem.");
        #typedef enum {False,True} Boolean;
        #typedef int  Integer;
        ##define Asub(I) A[(I)-1]    /* C's restriction that array*/
        ##define Bsub(I) B[(I)-2]    /* indices start at zero     */
        ##define Csub(I) C[(I)+7]    /* prompts definition of     */
        ##define Xsub(I) X[(I)-1]    /* macros to do subscripting.*/
        #                            /* Pascal equivalents:       */
                #static Boolean A[ 8];  /* A:array[ 1.. 8] of Boolean */
                        .data
0000            L00_DATA:
0000  00                .byte   0
                        .set    _A,L00_DATA+0
                #static Boolean B[15];  /* B:array[ 2..16] of Boolean */
0001                    .space  7
0008  00                .byte   0
                        .set    _B,L00_DATA+8
                #static Boolean C[15];  /* C:array[-7.. 7] of Boolean */
0009                    .space  15
0018  00                .byte   0
                        .set    _C,L00_DATA+24
                #static Integer X[ 8];  /* X:array[ 1.. 8] of Integer */
0019                    .space  15
0028  00                .byte   0
                        .set    _X,L00_DATA+40
                #void Try(Integer I, Boolean *Q) {
                        .text
0000                    .align  1
0000            L000:
                        .globl  _.Try
                _.Try:
0000  D961 FFB4         stm     r6,-76(r1)
0004  6E00              mr      r14,r0
0006  6D10              mr      r13,r1
0008  C811 FFB4         cal     r1,-76(r1)
000C  6C20              mr      r12,r2
000E  6B30              mr      r11,r3
                #   Integer J = 0;
0010  A4A0              lis     r10,0
                #   do {
                #      J++; *Q = False;
0012            L012:
0012  90A1              ais     r10,1
0014  A490              lis     r9,0
0016  109B              stcs    r9,0(r11)
                #      if (Asub(J) && Bsub(I+J) && Csub(I-J)) {
0018  C82E FFA8         cal     r2,-88(r14)
001C  682A              cas     r8,r2,r10
001E  CE38 FFFF         lc      r3,-1(r8)
0022  B439              c       r3,r9
0024  0A2D              je      L07E
0026  63AC              cas     r3,r10,r12
0028  6723              cas     r7,r2,r3
002A  4637              lcs     r3,6(r7)
MetaWare High C Compiler 1.3  07-Jul-86 17:13:14  queens.c  Page 4

                Eight Queens problem.
Addr  Object    Source Program and Assembly Listing
002C  B439              c       r3,r9
002E  0A28              je      L07E
0030  63C0              mr      r3,r12
0032  E23A              s       r3,r10
0034  6623              cas     r6,r2,r3
0036  CE36 001F         lc      r3,31(r6)
003A  B439              c       r3,r9
003C  0A21              je      L07E
                #        Xsub(I) = J;
003E  63C0              mr      r3,r12
0040  AA32              sli     r3,2
0042  E123              a       r2,r3
0044  39A2              sts     r10,36(r2)
                #        Asub(J) = False;
0046  DE98 FFFF         stc     r9,-1(r8)
                #        Bsub(I+J) = False;
004A  1697              stcs    r9,6(r7)
                #        Csub(I-J) = False;
004C  94C8              cis     r12,8
                #       if (I < 8) {
004E  89900016          bhex    L07A
0052  DE96 001F         stc     r9,31(r6)
                #            Try(I+1,Q);
0056  62C0              mr      r2,r12
0058  9021              ais     r2,1
005A  63B0              mr      r3,r11
005C  8DFFFFD2          balix   r15,_.Try       # Try
0060  C80E 0000         cal     r0,0(r14)
                #            if (!*Q) {
0064  402B              lcs     r2,0(r11)
0066  B429              c       r2,r9
0068  020B              jne     L07E
                #               Asub(J) = True;
006A  A491              lis     r9,1
006C  DE98 FFFF         stc     r9,-1(r8)
                #               Bsub(I+J) = True;
0070  1697              stcs    r9,6(r7)
                #               Csub(I-J) = True;
0072  89800006          bx      L07E
0076  DE96 001F         stc     r9,31(r6)
                #               }
                #            }
                #        else *Q = True;
007A            L07A:
007A  A421              lis     r2,1
007C  102B              stcs    r2,0(r11)
                #        }
                #      }
                #   while (!(*Q || J==8));
007E            L07E:
007E  402B              lcs     r2,0(r11)
0080  9420              cis     r2,0
0082  0203              jne     L088
0084  94A8              cis     r10,8
0086  02C6              jne     L012
0088            L088:
0088  61D0              mr      r1,r13
008A  C961 FFB4         lm      r6,-76(r1)
008E  E88F              br      r15
0090  DF07DF68          .long   0xDF07DF68
                                        # First gpr=r6
0094  2D00              .short  0x2D00  # npars=2, off=0
                        .data   1
```

```
MetaWare High C Compiler 1.3  07-Jul-86 17:13:14  queens.c  Page 5

                Eight Queens problem.
Addr  Object    Source Program and Assembly Listing
                        .globl  _Try
0058            _Try:
0058  00000000'         .long   L000
005C                    .align  2
                #   }
                #pragma Page(1); /* Page eject requested. */
                #void main () {
                        .text
0096                    .align  1
0096            L096:
                        .globl  _.main
                _.main:
0096  D9B1 FFC8         stm     r11,-56(r1)
009A  6E00              mr      r14,r0
009C  6D10              mr      r13,r1
009E  C811 FFC4         cal     r1,-60(r1)
                #   Integer I; Boolean Q;
                #   printf("%s\n","go");
00A2  C82E FFEC         cal     r2,-20(r14)
00A6  C83E FFF0         cal     r3,-16(r14)
00AA  8DF00000'         balix   r15,_.printf
00AE  CD0E 0004         l       r0,4(r14)
                #   for (I =  1; I <=  8; Asub(I++) = True);
00B2  A4C1              lis     r12,1
00B4            L0B4:
00B4  94C8              cis     r12,8
00B6  0B09              jh      L0C8
00B8  A421              lis     r2,1
00BA  6BC0              mr      r11,r12
00BC  E1C2              a       r12,r2
00BE  63BE              cas     r3,r11,r14
00C0  898FFFFA          bx      L0B4
00C4  DE23 FFA3         stc     r2,-93(r3)
                #   for (I =  2; I <= 16; Bsub(I++) = True);
00C8            L0C8:
00C8  A4C2              lis     r12,2
00CA            L0CA:
00CA  D40C0010          ci      r12,16
00CE  0B09              jh      L0E0
00D0  A421              lis     r2,1
00D2  6BC0              mr      r11,r12
00D4  E1C2              a       r12,r2
00D6  63BE              cas     r3,r11,r14
00D8  898FFFF9          bx      L0CA
00DC  DE23 FFAA         stc     r2,-86(r3)
                #   for (I = -7; I <=  7; Csub(I++) = True);
00E0            L0E0:
00E0  C8C0 FFF9         cal     r12,-7(r0)
00E4            L0E4:
00E4  94C7              cis     r12,7
00E6  0B09              jh      L0F8
00E8  A421              lis     r2,1
00EA  6BC0              mr      r11,r12
00EC  E1C2              a       r12,r2
00EE  63BE              cas     r3,r11,r14
00F0  898FFFFA          bx      L0E4
00F4  DE23 FFC3         stc     r2,-61(r3)
                #   Try(1,&Q);
00F8            L0F8:
00F8  A421              lis     r2,1
00FA  C8BD FFC7         cal     r11,-57(r13)
00FE  63B0              mr      r3,r11
```

```
MetaWare High C Compiler 1.3  07-Jul-86 17:13:14  queens.c  Page 6

                Eight Queens problem.
Addr  Object    Source Program and Assembly Listing
0100  8DFFFF80          balix   r15,_.Try       # Try
0104  CD0E 0008         l
                                r0,8(r14)
                #pragma Skip(3); /* Skip 3 lines. */
                #   if (Q)
0108  402B              lcs     r2,0(r11)
010A  9420              cis     r2,0
010C  0A11              je      L0012E
                #      for (I = 1; I <= 8;) {
010E  A4C1              lis     r12,1
0110            L00110:
0110  94C8              cis     r12,8
0112  0B0E              jh      L0012E
                #        printf("%4d",Xsub(I++));
0114  C82E FFF4         cal     r2,-12(r14)
0118  6BC0              mr      r11,r12
011A  90C1              ais     r12,1
011C  AAB2              sli     r11,2
011E  63BE              cas     r3,r11,r14
0120  CD33 FFC8         l       r3,-56(r3)
0124  8DF00000'         balix   r15,_.printf
0128  CD0E 0004         l       r0,4(r14)
012C  00F2              j       L00110
                #        }
                #   printf("\n");
012E            L0012E:
012E  C82E FFF8         cal     r2,-8(r14)
0132  8DF00000'         balix   r15,_.printf
0136  CD0E 0004         l       r0,4(r14)
013A  61D0              mr      r1,r13
013C  C9B1 FFC8         lm      r11,-56(r1)
0140  E88F              br      r15
0142  DF07DFB8          .long   0xDF07DFB8
                                        # First gpr=r11
0146  0D00              .short  0x0D00  # npars=0, off=0
                        .data   1
                        .globl  _main
005C            _main:
005C  00000096'         .long   L096
0060  00000000'         .long~
                                _printf
0064  00000058'         .long   _Try
0068                    .align  2
                        .data
0029                    .space  31
0048            .LITERALS.:
0048  25730A            .ascii  "%s\012"
004B  00                .byte   0
004C  676F              .ascii  "go"
004E  00                .byte   0
004F                    .space  1
0050  253464            .ascii  "%4d"
0053  00                .byte   0
0054  0A                .ascii  "\012"
0055  00                .byte   0
                        .data
0056                    .space  2
No user errors   4 unprinted warnings
End of processing, 07-Jul-86 17:13:19          queens.c
```

10. MAKING CROSS REFERENCES

This chapter explains how to use the **hcxref** command to generate a cross-reference listing of one or more High C modules.

10.1. Features of the Cross Reference

Cross references have the following features:

- *References to source files.* All cross-reference information refers to line numbers within files compiled, as opposed to line numbers within a listing. Therefore no listing is necessary to use the cross reference.
- *Include files.* Included source files are handled properly. That is, they do not interfere with the process, and their names are included correctly in the results.
- *Assignments versus uses.* References that assign values into variables are distinguished from references that use values of variables.
- *Annotated listing.* It is possible to generate an annotated source listing of one or more program files. The listing contains cross-reference information to the right of the source text listed.
- *Multi-module cross references.* A cross reference can span multiple compilation units by cross-referencing many modules at once and showing references from one module into the other. Thus, a single cross reference can be produced for a program that is broken up into separately compiled modules.
- *Inter-module usage summaries.* A list of the names that one module uses that are located in other files can be produced, organized by file. This helps one understand the module interconnectivity of a large program.

10.2. Using the **hcxref** Command

The **hcxref** command processes one or more High C source files and produces a cross-reference listing on standard output. The listing consists of up to four components as described in §10.3 below.

The command has the following form:

```
hcxref  [-ilmpus] [preprocessor_options]... files...
```

where *files* denotes one or more High C source files, and *preprocessor_options* denotes zero or more preprocessor options (e.g. -I*dir* or -D*name*) that are required when compiling the files.

The **-l** option causes a listing of the source files to be generated, annotated with cross-reference information. Include files are not expanded in the listing unless **-i** is also specified.

The **-m** option causes a listing to be produced, for each module M, of the names referenced in M that were defined elsewhere.

Names that are declared but not referenced do not appear in the cross reference unless the **-u** option is specified.

The **-p** option causes the outboard C preprocessor to be invoked on each source file. The output of the preprocessor is then processed by the cross referencer instead of the source files themselves. If this option is not specified, the inboard preprocessor is used. This option is analogous to the **-Hcpp** option of the **hc** command. The **-l** and **-i** options are ignored when used in conjunction with **-p**.

The **-s** option specifies that various statistics relating to the cross reference are to be printed.

The **hcxref** command invokes the High C compiler in a special mode to generate the cross-reference information. Therefore, if any of the source files contains errors, appropriate diagnostics are generated.

10.3. Cross-Reference Format

Components. Each cross reference is self-documenting and consists of four components:

(1) An alphabetized list of all names declared in the program, together with an ordered list of all the references to each name.

(2) An alphabetized table of all files used in the program and a file reference number for each.

(3) A list for each module `M` of all the names used by `M` that are declared in other files — if requested.

(4) An annotated cross reference for each module — if requested.

When the components are produced:

Item (1) is always produced.

Item (2) is produced if the cross reference involves more than one file; this happens if more than one module is cross-referenced, or if any compiled include files were involved in the modules being cross-referenced.

Item (3) is produced if the **-m** option is specified.

Item (4) is produced if the **-l** option is specified.

What each component consists of:

Item (1) presents the following information for each distinct name in the program:

- The line and column number of the declaration of the name. If the name occurs in a compiled include file, or if several modules are being cross-referenced, the file number is also given.
- The declared name `N`, and its *owner*: the name of the function that contains `N`'s declaration.
- Information about the named object, such as its storage class (**static, extern, typedef, register**, etc.) and in some cases, the object's type.
- The numbers of any lines containing references to the name. If the references are not in the module being cross-referenced, such as in an include file, or if several modules are being cross-referenced, the line numbers are presented in the format "`fn<I>`" where `n` is the number of the file containing the references and "`I`" is the list of line numbers. Occasionally the entry in this field is of the form "`resolved at` *`ref`*" where `ref` is a line number or `fn<...>` reference as just described. This means that the name was introduced by an **extern** declaration whose actual definition was given at `ref`.
- References that assign, or may assign, a value to a variable are marked with the character "*".

Item (2) presents the correspondence between file numbers and file names. References in items (1) and (3) use the file number rather than file names, to keep the listing brief. Use item (2) to determine the corresponding file name.

Item (3) is optional. It is requested by the **-m** option. The output produced is a listing for each module `M` of the names used by `M` that are declared in other files. The list is organized by file. This is useful for determining the interconnectivity between modules.

In Items (1) and (3) a reference to a name `N` declared at reference point `P` is changed to a reference to a point `P'`, if the definition at `P'` resolves the declaration at `P`. Typically this will happen when `N` is declared in an interface file `F`, is used from a module `M`, and is defined at `P'` in a module `M'`. The module usage in Item (3) will show that `M` refers to `P'` in module `M'`, *not* `P` in interface file `F`. That is, one gets references to the implementations rather than the interfaces through which they were supplied.

Item (4) is optional and is is generated by the `-l` option. The result is a line-numbered listing of the source of the program compiled, with each line annotated on the right with the line numbers where the names used on the line were defined.

If n names are used on the line, n line numbers appear to the right of the line, corresponding positionally. A line number alone is a reference into the file being listed. If the letter "`i`" appears instead, the name referenced is an intrinsic, such as "`_find_char`" or "`_abs`". Finally, a line number followed by "`f`" and another number means that the name was declared in a file other than the one being listed; the file number can be used to discover that file's name in Item(2). "`Line#fFile#`" was used instead of "`File#<Line#>`" as in Item (1) for brevity.

10.4. Distinction of File Names

In a multi-module cross reference, a particular interface file may have been included by several modules because each of the modules being cross-referenced needs the resources in that file. The cross-referencer assumes that a repeated declaration of a name in a compiled include file is the same declaration if it appears at the same line and column number of the same include file.

For purposes of determining "sameness of include files" the cross referencer uses the text of the file name including the path. Therefore to cross-reference several modules successfully, do not use different names for the same include file.

For example, if module `M1` includes "`../utils/trees.h`" and `M2` includes "`/prog/utils/trees.h`" and if these two references denote the same file, the cross referencer will *not* recognize them as the same.

11. SYSTEM SPECIFICS

This section contains some system-specific aspects of the High C compiler on 4.2/RT.

11.1. Floating-Point Arithmetic

High C uses the IEEE Standard 754 formats to represent floating-point data.

Each **float** is a 32-bit value with an 8-bit exponent and a 23-bit mantissa. The absolute values of the representable numbers lie in the range 8.43×10^{-37} to $3.37 \times 10^{+38}$.

Each **double** and **long double** is a 64-bit value with an 11-bit exponent and a 52-bit mantissa. The absolute values of the representable numbers lie in the range 4.19×10^{-307} to $1.67 \times 10^{+308}$.

11.2. Input Line Length

The default maximum input line length for the compiler is 2,000 characters.

11.3. Some ANSI-Required Specifics

Here are some additional system specifics that the ANSI document X3J11/86-102 requests each C implementation provide.

Identifiers. The number of significant characters in an identifier is 2,000, since that is the longest input line acceptable to the compiler. Identifiers that are global appear in the object-module name list with an underscore prefix. Casing is preserved.

Characters. The characters in the source and the execution character set are the standard ASCII characters. Each character in the source character set maps into the identical character in the execution character set. Without exception, all character constants map into some value in the execution character set.

A character is stored in a byte and there are four bytes in an **int**.

High C does not permit a character constant that contains more than one character. Such a construction is usually machine-dependent.

The type specifier **char**, when not accompanied by an adjective, denotes type **unsigned char**. However, this can be changed by turning `Off` the toggle `Char_default_unsigned`.

Integers. Integers are represented in twos-complement binary form. The following table illustrates the ranges of values to which the various integer types are restricted:

Type	Range		
signed char	-128	to	127
unsigned char	0	to	255
short	-32,768	to	32,767
unsigned short	0	to	65,535
int	-2,147,483,648	to	2,147,483,647
unsigned int	0	to	4,294,967,296
long	-2,147,483,648	to	2,147,483,647
unsigned long	0	to	4,294,967,296

Conversion of an integer to a shorter signed integer or **int** bit field is done by bit truncation; i.e. when storing an X-bit value into a Y-bit receptacle, where X > Y, the rightmost Y bits of the first value are stored. Conversion of an unsigned integer U to a signed integer I where **sizeof**(U) = **sizeof**(I) consists in transferring the bits of U into I, whether or not the value of U is representable in I. For example, (**short int**)(**short unsigned**)65535 is the **short int** value `-1`.

The results of bitwise operations on signed integers are the same as if the integers were treated as unsigned.

The sign of the remainder on integer division is the same as the sign of the dividend.

The right shift of a signed integral type is arithmetic; i.e. the sign bit is propagated to the right.

Floating point. Floating-point representation is IEEE Standard 754. The default rounding mode is "round to nearest". See §5 *STORAGE MAPPING* for the length required for each floating-point type.

When a negative floating-point number is truncated to an integral type, the truncation is toward zero. Thus `-2.7` is truncated to `-2` and `-1.2` to `-1`.

Registers. **`register`** class variables of the following types are considered eligible for assignment to registers: signed or unsigned integer of any size, **`float`**, **`double`**, and pointer.

A **`double`** can be mapped only to a floating-point register. A **`float`** can be mapped to either a floating-point or a general register. All others can be mapped only to general registers.

Potentially, as many variables can be placed in registers as there are "non-volatile" registers. See §6 *RUN-TIME ORGANIZATION* for a list of the non-volatile registers.

Structures, unions, and bit fields. Only unsigned bit fields are currently supported. A bit field declared as **`int`** is treated as **`unsigned int`**. For more information on structures, unions, and bit fields, see §5 *STORAGE MAPPING.*

Declarators. There may be at most 65,000 declarators modifying a basic type.

Statements. There may be at most 65,535 cases in a **`switch`** statement.

Preprocessing directives. A single-character constant in a constant expression controlling conditional inclusion is always positive in value, ranging from 0 to 255.

`#pragma` directives are neither recognized nor permitted.

For the method of locating includable source files, see §2 *INVOKING THE COMPILER.*

12. DIAGNOSTIC MESSAGES

Messages from the High C compiler report: (a) file I/O errors, (b) system errors, and (c) user errors and warnings.

12.1. File I/O Errors

File I/O errors are fatal. They can occur in attempting to open a non-existent file or in writing a compiler output file when not enough file space is available. The errors likely to be seen are:

`Unable to open file fff: file not found.`

This message is produced when any input source file, such as that specified on the command line or in an `Include` pragma, cannot be found.

This message is produced twice: it is written once to standard output and once to standard error. If standard output is not redirected, the message appears on the screen twice.

```
***Error occurred on writing instruction file: ...
***Error occurred on writing object file: write failed.
***Write error occurred during tree paging.
```

Usually caused by too little space on disk. Remove unnecessary disk files and try again.

Note: Fatal errors may result in compiler temporary files being left in the "`/tmp`" directory. They should be removed.

12.2. System Errors

System errors are fatal and should rarely occur. They take the following form:

`>>>>> S Y S T E M   E R R O R  n <<<<<, in` ***`Module:Function`***
Error message text.

where **`n`** numbers the occurrences of system errors, **`Module`** is the module name, and **`Function`** is the function name. The only system error messages that the user should be concerned with are:

```
Dynamic array allocation/reallocation failed.
Out of memory.
```

This error indicates that either a dynamic array in the compiler exceeded 65,535 elements or the user's virtual memory quota was exceeded. The former case should not occur unless the source program is pathologically large.

`Exceeded Card_char_limit.`

An input line was encountered that exceeded the limit of 2,000 characters.

`Recover: Exceeded the following limit:` ***`Limit.`***

In repairing a syntax error, a table overflowed. The table limit is fixed, so no increase in memory can improve the situation. Repair the syntax error.

There are many other system error messages that the compiler could produce, but they are associated with internal compiler errors or inconsistencies that should not occur.

Note: Fatal errors may result in compiler temporary files being left in the "`/tmp`" directory. They should be removed.

Stack dump. Compiler system errors are always accompanied by a call-stack dump. The dump can usually be ignored, but when reporting a problem to the support staff, the history of called functions can be helpful; include a listing of the dump in any written correspondence. The following is a sample dump.

```
>>>>> S Y S T E M   E R R O R 1 <<<<<, in Scanner:Read_scan_tables
                                           No scan tables found.
                                    Line
Routine                File         /Off Addr  Parms...
SYSERR                 syserr.p     66   54d3a c098,c080,0,66290,66320
READ_SCAN_TABLES       stread.p     69   bef2  2004a4c,fffa60,44ed,663c4
GET_SCAN_TABLES.STREAD stread.p     39   c005  663c4,66290,fffadc,14e8,1
ANALDRVR               analdrvr.p 19     44ed  1,663c4,66416,0,1,186dc
INITIALIZE_PREFIX.SK   skelinit.p 2dc    14e8  fffaec,115a,fffaf4,59645
DOIT                   skeldrvr.p e      111f  fffaf4,59645,fffb04,4d,3
pp_MAIN                skeldrvr.p 6      115a  fffb04,4d,3,fffb08,fffb18
_main                  ppinit       1d   59645 3,fffb08,fffb18,ufffb38
start                               3d   4d    fffb3d,fffb45,0,fffb4b
Error was severe.  Program terminated.
```

The **Routine** and **File** columns are usually sufficient alone when reporting a problem to support personnel.

System errors due to a bug in the compiler's code generator are accompanied by a line "`Code was being generated for program text near Ln/Cm.`" following the call-stack dump. This helps isolate the program text causing the problem and may facilitate reducing the problem program to a few lines, which then can be easily sent to compiler support personnel.

NOTE: When code generator errors occur, they can frequently be "cured" by inserting a label before the line causing the problem. Even if this cures the problem, please report the problem to support personnel.

12.3. User Errors and Warnings

User error messages are grouped in the three categories: (1) lexical, (2) syntactic, and (3) constraint. Warnings do not suppress object file generation; errors always do. Also, some diagnostics that are warnings become errors when the compiler is run in ANSI mode.

Messages that report errors terminate compilation after the phase issuing the diagnostic, so errors that would otherwise have been detected by later phases are not reported until the earlier error is repaired and the compiler is re-invoked.

All user diagnostics are accompanied by the file name, a line number **n**, and column number **m**, where the error was detected, in the form `"filename", Ln/Cm`. In addition when **-Hlist** is specified on the command line as assumed in the examples below, lexical and syntactic errors are generally accompanied by the erroneous line with a caret ^ beneath it at the point of error detection. (Errors begin with "E" and warnings with "W", and usually occupy a single line.)

Lexical error messages are produced when an improperly formed word is detected, such as a string with a missing closing quote. ***Example***:

```
Levels  LINE # |----+----1----+----2----+----3----+----4----+----5
             1 |void main() {
1 1          2 |   char *S;
1 1          3 |   S = "Hello;
           C15 ------------- -^
E "file", L3/C15:  (lexical)  Unexpected end-of-line encountered.
1 1          4 |   }
```

Syntactic error messages are produced for programs that are ill-formed on the phrase level, such as a missing "`;`" or inserted spurious symbol. The message is accompanied by a statement of the `REPAIR` that the compiler effected so it could keep processing input. ***Example***:

```
Levels  LINE # |----+----1----+----2----+----3----+----4----+----5
             1 |void main() {
1 1          2 |   printf "Hello");
           C11 ----------^
1 1          3 |   }
E "file", L2/C11:  (syntactic)  unexpected symbol:'<STRING>':"Hello"
REPAIR:   '(' was inserted before '<STRING>':"Hello"@L2/C11
```

Constraint error and warning messages diagnose more subtle problems, such as an undeclared identifier or type mismatch. There are nearly 200 such diagnostics, each of which is meant to be self-explanatory. Most of them prevent the generation of object code, but some are merely warnings and are intended to assist the programmer. ***Example:***

```
Levels  LINE # |----+----1----+----2----+----3----+----4----+----5
             1 |void main() {
1 1          2 |   int i;
1 1          3 |   i = Undeclared_identifier;
1 1          4 |   }
E "file", L3/C8:   Undeclared_identifier: This is undeclared.
1 user error    No warnings    453K of memory unused.
```

```
Levels  LINE # |----+----1----+----2----+----3----+----4----+----5
             1 |void main() {
1 1          2 |   int i, Unused;
1 1          3 |   i /= 0;
1 1          4 |   }
w "file", L2/C8:   i: Variable is set but is never referenced.
w "file", L2/C11:  Unused: Variable is never used.
E "file", L3/C6:   Division by zero.
1 user error   2 warnings    457K of memory unused.
```

12.4. Error and Warning Messages

This section presents all compiler diagnostic messages, except automatically generated lexical and syntactic messages, in alphabetical order, with explanations where appropriate.

`"=" used where "==" may have been intended.`
"=" was detected as an operator in a Boolean expression, such as "`if (x = y) (...)`". Often this is a mistake, as "`if (x == y) (...)`" was intended.

`"auto" must appear within a function.`
Storage class **auto** cannot be given for declarations that do not appear within a function.

`"break" must appear within while, do, for, or switch.`

`"case" must appear within a "switch".`

`"continue" must appear within while, do, or for.`

`"default" must appear within a "switch".`

`"pragma Data" active at end of module.`

`"pragma Data" active at end of function.`
A "`pragma Data(...);`" was given in a module or function, with no terminating "`pragma Data;`". This is permitted but the programmer may have forgotten to supply the terminating pragma, thus perhaps including more data declarations in a data segment than intended.

`"register" is the only allowable storage class for a parameter.  Ignored.`
In a function definition or declaration, a storage class other than **register** was given, such as in "`int f(i) static i; {...}`".

`"register" must appear within a function.`
Storage class **register** cannot be given for declarations that do not appear within a function definition.

`"void" is illegal here.`

`A bit field is not valid as an argument to &.`
One cannot take the address of a bit field, since such a field is not necessarily on a byte boundary.

`A bit field is not valid as an argument to` **`sizeof`**`.`
Since bit fields need not occupy an integral number of bytes, taking their **`sizeof`** is prohibited.

`A function may not return a function (but may return a pointer thereto).`

`A function may not return an array (but may return a pointer thereto).`

`A function may not return an incomplete type.`
A function cannot return a **`struct`** or **`union`** type whose fields have not yet been specified. For example, "**`struct`** `s;` **`struct`** `s *f() {...}`" is legal since `f` returns a *pointer* to an incomplete **`struct`** type, but "**`struct`** `s;` **`struct`** `s g() {...}`" is illegal.

`A functionality` **`typedef`** `cannot be used in a function definition.`
"**`typedef int`** `f(); f g {`**`return`** `3;}`" is illegal: the type definition for `f` cannot be used to specify that `g` is a function.

`A parameter may not be a function (but may be a pointer thereto).`

`A parameter name must be given here.`
For function definitions, parameter names must be supplied. Thus, for example, "**`void`** `f(`**`int`**`,` **`float`** `g) {...}`" is illegal because the first parameter lacks a name.

`A register-class function makes no sense.`
For example, "**`register`** `f() {...}`" is illegal.

`An array may not contain functions (but may contain pointers thereto).`

`An array must have a positive number of elements.`

`An array of objects of an incomplete type is illegal.`
An array cannot contain a **`struct`** or **`union`** type whose fields have not yet been specified. For example, "**`struct`** `s;` **`struct`** `s *a[10];`" is legal since "a" contains *pointers* to an incomplete **`struct`** type, but "**`struct`** `s;` **`struct`** `s b[10];`" is illegal.

`An object of type` ttt `cannot be initialized.`

`Argument to "`**`#include`**`" must be a string.`

`Argument type` ttt `is not compatible with formal parameter type` ttt'.
An attempt was made to pass an argument of a wrong type to a function, such as passing a **`float`** for a parameter that is a **`struct`**. When using standard C function definitions, this is a warning only, since C permits such mismatches; but when using prototype syntax, it is an error. This warning provides the security of Pascal function call semantics.

`Array size exceeds addressability limits.`

`Bit fields must fit in a register or register pair.`

`Cannot dereference a pointer to` **`void`**`.`
Type **`*void`** was introduced as a means of defining a "generic pointer" compatible with other pointers. But there is no such thing as an object of type **`void`**. Therefore, dereferencing a pointer to **`void`** is illegal.

`Cannot initialize a` **`typedef`**`.`
Something like "**`typedef int`** `T = 1;`" was attempted.

`Cannot initialize an imported variable.`
Something like "**`extern int`** `T = 1;`" was attempted. A variable may be initialized only by its definition.

`Cannot take` **`sizeof`** `a function type.`

`Cannot take sizeof an incomplete type.`

The **`sizeof`** a **`struct`** or **`union`** type whose fields have not yet been specified is not known. For example, what follows is illegal since the size of the structure is unknown: "**`struct`** `s; (...)` **`sizeof`**`(`**`struct`** `s)(...)`".

`Cannot take sizeof type void.`

There are no objects of type **`void`**, therefore taking **`sizeof void`** makes no sense.

`Cannot take the address of a register variable.`

`Declared type is never referenced.`

`Divide by zero.`

This was detected in a constant expression at compile time.

`Enclosing function's return type is "void"; therefore nothing may be returned.`

"**`return`** `E;`" for some expression `E` was found in a function whose return type is **`void`**.

`End of file encountered within #if construct.`

`End of file encountered within arguments to a macro. Probably a missing right parenthesis.`

`End of file encountered within macro definition.`

`End of file encountered within macro formal parameter list.`

`Expression has no side effect and has been deleted.`

An expression used in a statement context has no side effect; therefore the expression is useless. For example, "`2+3;`".

`External function is never referenced.`

`Fewer arguments given than function has parameters.`

`for` `loop will never execute.`

`Function called but not defined.`

Any function that was called but not defined is noted as a warning. Although such practice is permissible in C, especially useful when calling library functions, a common error is to misspell a function name. The error goes undetected until link-time without this warning. Furthermore, errors in parameter linkage can occur when a call is made to an undefined function. We recommend that the library "`.h`" header files always be included to get parameter checking, and that function prototypes be used for external function declarations, rather than making use of the "feature" of C for calling undefined functions.

`Function expected.`

The expression `f` preceding the arguments in a function call `f(...)` must denote a function.

`Function parameter names are allowed only on function definitions, not declarations.`

"**`int`** `f(a,b,c);`" is a function declaration that names the parameters `(a,b,c)`. This is illegal unless function prototype syntax is used, as in "**`int`** `f(`**`int`** `a,` **`int`** `b,` **`int`** `c);`".

`Function return value never specified within function.`

A function with a non-**`void`** return type contains no **`return`** statement. This typically happens with "old" C programs that did not use **`void`** to indicate that a function returns nothing.

`Functions may not be nested.`

In ANSI-Standard C, functions cannot be declared within functions. In High C they can. This message is produced when the compiler is doing ANSI checking.

`Identifier required after #ifdef or #ifndef.`

`Identifier required. Pragma ignored.`

`Incompatible tag reference: The` **ttt** `tag class does not match the tag class` **ttt'** `defined at Ln/Cm.`

Something like "**struct** s; **union** s {**int** x;};" was encountered. The tag s cannot simultaneously be the tag for a **struct**, **union**, and/or **enum**.

`Incomplete type: the` **struct/union** `type at Ln/Cm must be completed before it can be used here.`

A reference has been detected to a field of a **struct** or **union** type whose fields have not yet been specified.

`Incorrect number of parameters to macro. Macro invocation ignored.`

The number of arguments to a macro must agree exactly with the number of parameters in its **#define**.

`Integer constant exceeds largest` **unsigned** `number.`

`Invalid digit in non-decimal number: X.`

`Local function is never referenced; no code will be generated for it.`

A function of storage class **static** is not called anywhere in the compilation unit. Since it is not exported, there can be no reference to the function and it is essentially deleted.

`Lower bound of range is greater than upper bound.`

This can only happen in High C **case** statements where range expressions are allowed as labels (an extension). Macro name must be an identifier.

`Macro parameter must be an identifier.`

`Members cannot be of an incomplete type.`

A **struct** or **union** cannot contain a **struct** or **union** type whose fields have not yet been specified. For example, "**struct** s; **struct** t {**struct** s *p;}" is legal since p is a *pointer* to an incomplete **struct** type, but "**struct** s; **struct** t {**struct** s p;}" is illegal.

`Mismatched` **#if-#elif-#else-#endif.**

`More arguments given than function has parameters.`

`Must be a compile- or load-time computable expression.`

The initializers for a static variable must be determinable when a program is loaded.

`Must be a compile-time computable constant.`

`Must be a pointer.`

`Must be a scalar (`**int**`,` **char**`, floating, or pointer) type.`

`Must be a static variable reference.`

`Must be a string.`

`Must be a` **struct** `or` **union**`.`

`Must be a type.`

`Must be an identifier.`

`Must be an integral` **int** `or` **char** `type.`

`Must be of a pointer type.`

`Must be of an extended-function type.`

`Named parameter association is prohibited for this function since its declaration near Ln/Cm does not name all parameters.`

An attempt was made to call a function F using named parameter association, but F's declaration did not name all of its parameters. For example,

```
void F(int a,float);  (...)F(a=>37,   3.3);/*Illegal.*/
void F(int a,float b);(...)F(a=>37,b=>3.3); /*Fine.*/
```

`No "pragma Data" is active.`

"**pragma** `Data;`" was encountered without a preceding, and matching, "**pragma** `Data(...);`".

`No member is declared here.`

A declaration with no declared object was found within a **struct** or **union**. For example,

```
struct s {int; float; struct t {int y};}
```

contains three declarations, none of which declare an object. However, this construct is not entirely vacuous because the declaration of **struct** `t` is visible outside of **struct** `s` and therefore can be used to declare objects of type **struct** `t`.

`No object may be of type void.`

`No parameter declarations may be given here.`

In defining a function using prototype syntax, where the parameter types were specified in the parameter list, an attempt was made to re-declare the parameters following the parameter list. For example, "**int** `x,y;`" is illegal in "**void** `f(`**int** `x,` **int** `y)` **int** `x,y; { ... }`".

`Non-decimal constant exceeds largest unsigned number.`

`Only a parameter may be declared here.`

Preceding a function definition's {, only the function's parameters may be declared.

`Only fields of type "unsigned int" or "unsigned long int" are supported.`

Bit fields may be of only these two types. Any bit field of another type is coerced to one of them, depending upon the size of the bit field.

`Only one "default" is permitted in a "switch".`

`Operand type inappropriate for operator.`

An inappropriate operand was detected for a built-in operator such as `&`, `|`, `~`, etc. For example, "**float** `f1,f2; (...)f1 = f1 & f2;`" is illegal: & requires integral operands.

`Parameter not found or specified more than once.`

In a function call using named parameter association, a parameter was named twice, or a non-existent parameter was referenced.

`Parameter ppp not supplied.`

In a function call using named parameter association, parameter ppp was not given an argument value.

`Parameter separator must be a comma.`

In a **#define** of a macro with parameters, parameter names must be separated by commas. For example, "**#define** `M(a b) c`" is illegal; "`a,b`" is required.

`Pointer dereferencing disallowed in static context.`

`Pragma has too few parameters.`

`Pragma has too many parameters.`

`Previous "pragma Data" is still active.`

"**pragma** `Data(...);`" was given in the context of an already active "**pragma** `Data(...);`". Insert "**pragma** `Data();`" preceding the offending pragma to "turn off" the active pragma.

`Real constant has too many digits.`

`Result of comparison never varies.`

An expression was found whose operands, while they are not all constants, are such that the value of the expression is always the same. For example, an expression of type **unsigned int** is always less than zero.

`Right operand of shift operator is negative.`

`Since the first parameter was specified by the type "`**`void`**`", there may be no other parameters.`

The special syntax exemplified by "**`int`** `f(`**`void`**`);`" denotes a function `f` taking no parameters. Because of this, no parameter can be specified after "**`void`**": "**`int`** `f(`**`void`**`, `**`float`**`, `**`int`**`);`" is illegal.

`Size change in cast involving pointer type: casted-to type` ttt `is not the same size as casted-from type` ttt′`.`

`Specified storage class for this declaration is unnecessary and was ignored.`

In a declaration such as "**`static struct`** `s{`**`int`** `x;};`", the storage class "**`static`**" is useless since no object was declared.

`Static initialization of bit fields is not supported.`

`Storage-class nonsensical for function definition.`

`String too long for initialized array.`

`Structure has no contents (is of size zero).`

`Subscripted expression must be an array or pointer.`

`The 2nd and 3rd operands of a conditional expression must be both arithmetic, or of the same type, or one a pointer and the other zero.`

`The declarator must be a function. This declaration has been discarded.`

A declaration such as "**`int`** `f {...};`" was encountered, where a function body `{...}` was given for a non-function.

`The rest of this line is extraneous.`

`The sign (`**`signed/unsigned`**`) has been specified more than once.`

`The storage-class (`**`auto`**`, `**`extern`**`, etc.) has been specified more than once.`

`The width (`**`long/short`**`) has been specified more than once.`

`This "`**`return`**`" should return a value of type` ttt `since the enclosing function returns this type.`

`This can be of an incomplete type only if it is "`**`extern`**`" or has an initializer supplying its size.`

`This code will never be executed.`

`This construct would have been deleted as an optimization had it contained no labels.`

A construct such as "**`while`** `(0) {...}`" was detected but cannot be deleted due to the presence of one or more labels within `{...}`. This is questionable programming practice at best.

`This function declaration is inconsistent with the "`**`int`**`"-returning function declaration imputed at Ln/Cm.`

A function called before it is declared is assumed to be a function returning **`int`**, and any subsequent declaration of the function must declare it to be so. For example, "`main () { (...) f(3);(...) }` **`void`** `f() {...}`" is illegal since `f` was called before being defined and therefore assumed to return **`int`**.

`This function declaration is inconsistent with the declaration at Ln/Cm.`

`This is already defined as a macro. Redefinition ignored.`

A redefinition of a macro is permitted only if the redefinition agrees exactly with the previous definition. To otherwise redefine a macro, use **`#undef`** to explicitly undefine the macro before re-defining it.

`This is multiply declared.`

`This is permissible only in conjunction with "`**`int`**`" or "`**`char`**`".`

`This is permissible only in conjunction with "`**`int`**`" or "`**`double`**`".`

`This is permissible only in conjunction with "`**`int`**`".`

`This is undeclared.`

`This may not be a pointer to a function (but may be a pointer to an object).`

`This tag name is more than 80 characters long.`

`This type lacks a tag and hence cannot be used.`
A declaration such as "**`struct`** `{`**`int`** `x;};`" was encountered. Without a tag the **`struct`** cannot be referenced and hence is useless.

`Toggle name required. Pragma ignored.`

`Too many initializers here.`

`Type` ttt `is not assignment compatible with type` ttt'`.`
(a) In an assignment expression, the right operand of type ttt may not be assigned to the left operand of type ttt'.

(b) In a function call, an argument of the type ttt may not be passed to a function that expects a parameter of type ttt'.

`Type` ttt `is not compatible with type` ttt'`.`
In a comparison, the left operand of type ttt may not be compared with the right operand, of type ttt'.

`Unexpected symbol in expression. Line ignored.`

`Unknown preprocessing directive.`

`Unrecognizable Data class. Static assumed.`

`Unrecognizable field name.`

`Unrecognizable pragma name. Pragma ignored.`

`Unrecognizable toggle name. Pragma ignored.`

`Up-level reference to a `**`register`**`-class variable is not allowed.`

`Variable is never used.`

`Variable is referenced but is never set.`

`Variable is set but is never referenced.`

`Variable is referenced before it is set.`

`Variable required.`
In this context a so-called "lvalue" is required but was not found. An *lvalue* is something whose address can be taken, and is required on the left side of an assignment expression and as an operand to `&`, `++`, and `--`. The rules of C require the automatic conversion of some objects into non-lvalues. For example, an lvalue of type array-of-T is always converted to a (non-l)value of type pointer-to-T, so it is *never* allowable to take the address of an array. So, "**`int`** `a[10]; (...) f(&a);`" produces the "`Variable required.`" diagnostic due to the application of `&` to "`a`", which has been converted to the address of the first element. Remove the `&`.

`Zero-length bit fields may not be named.`
A declaration such as "**`struct`** `{`**`int`** `i:0,j:2};`" was encountered. "`i`" must be omitted. As is, it is possible to refer to the field. Such a reference would be illegal.

`{...} inappropriate here for initializing a scalar.`

Appendix A. CROSS-JUMPING OPTIMIZATIONS

MetaWare compilers support a major optimization that usually obtains a 2-5% reduction in code size and is often accompanied by a decrease in execution time. The optimization is known as "cross-jumping". It, along with the two toggles that control it, is explained here.

Consider the following source code:

```
            if (!eof) readbytes(&buf,&cnt,512);  /* Code C. */
/* L: */    while (cnt > 0) {
               writebytes(&buf,cnt);
               if (!eof) readbytes(&buf,&cnt,512); /* Code C'.*/
               }  /* Implicit jump back to the implicit label.*/
```

The compiler can improve the code size of this program without any loss in execution speed by effectively re-writing the code as:

```
Top:        if (!eof) readbytes(&buf,&cnt,512);
                                                /* Code C = C'. */
/* L: */    if (cnt > 0) {
               writebytes(&buf,cnt);
               goto Top;
               }
```

The optimization involves the recognition of some code `C` immediately preceding a jump j to some label `L`, where some code `C'` identical to `C` immediately precedes `L`. The transformation consists in deleting `C` and replacing j with a jump to `C'` instead:

```
          some code C              jmp L'
          jmp L          =>
          ...                      ...
          some code C'       L':   some code C = C'
    L:    ...                L:    ...
```

This optimization is called "cross jumping" or "tail merging" in the compiler literature, since it was first invented to handle common code at the ends of the arms of conditional statements, and was effected by jumping across from one arm to the other, i.e. by merging the tails of the two arms. It is surprisingly effective and always saves code space while never giving up execution speed.

Here we include another optimization under that name as well. The second optimization is even more effective but gains (sometimes considerable) code space in trade for a small loss of speed. Consider the program fragment

```
if          (buf[cnt]==0) g(&buf);
   else if (buf[cnt]== '\n') {buf[cnt] = 0; g(&buf);}
   else ...
```

The compiler effectively transforms this into

```
if          (buf[cnt]==0) goto L';
   else if (buf[cnt]=='\n') {buf[cnt] = 0; L': g(&buf);}
   else ...
```

Here, both occurrences of "`g(&buf);`" precede a jump to the statement following the entire conditional. One of the instances of "`g(&buf);`" is replaced with a jump to the other, saving the code space for the call to `g` at the expense of inserting an additional jump. Opportunities for this kind of optimization are even more frequent than the standard cross-jumping optimization. In general the optimization can be depicted as follows:

```
         some code C          jmp L'
         jmp L
         ...                  ...
         some code C'    =>   L': some code C = C'
         jmp L                jmp L
         ...                  ...
    L:   ...             L:   ...
```

Both optimizations are turned `On` by default. Both may be disabled by turning `Off` the toggle `Optimize_xjmp`, with either "**-Hoff**=`Optimize_xjmp`" on the compiler execution line, or including "**pragma** `Off(Optimize_xjmp);`" in the program. The second of the two optimizations can be disabled by turning `Off` the toggle `Optimize_xjmp_space`, so named because the second optimization saves space but always increases execution time.

During the development phase of a project it may be desirable to turn `Optimize_xjmp Off`. The reason is that the optimization can cause such a contortion of code that using debuggers, whether assembly-language level or line-oriented symbolic, is difficult. As a case in point consider the following program, which compares the fields of two different structures to see if they are the same:

```
union {
    struct {int x,y;}           f1;
    struct {int a,b,c;}         f2;
    struct {int e,f;}           f3;
    struct {int g,h; int i[10];} f4;
    } u1,u2;

int f(i) int i; {
    switch(i) { /* What kind of structure to compare? */
        case 1: return u1.f1.x == u2.f1.x &&
                       u1.f1.y == u2.f1.y;
        case 2: return u1.f2.c == u2.f2.c &&
                       u1.f2.a == u2.f2.a &&
                       u1.f2.b == u2.f2.b;
        case 3: return u1.f3.e == u2.f3.e &&
                       u1.f3.f == u2.f3.f;
        case 4: return u1.f4.g == u2.f4.g &&
                       memcmp(u1.f4.i,u2.f4.i,
                        sizeof(u1.f4.i)) !=0;
        case 5: return u1.f4.h == u2.f4.h &&
                       memcmp(u1.f4.i,u2.f4.i,
                        sizeof(u1.f4.i)) !=0;
    } }
```

Here cases 1 and 3 are recognized as being identical, and matching the tail end of case 2. Furthermore cases 4 and 5 share a common tail. Compiling the code produces the following tightly-coded result that surpasses the usual patience of even a skilled assembly-language programmer in optimizing:

```
        .globl  .oVncs
        .set    .oVncs,0
        .globl  _memcmp
        .globl  _.memcmp
        .comm   _u1,48
        .comm   _u2,48
#union {
#       struct {int x,y;}           f1;
#       struct {int a,b,c;}         f2;
#       struct {int e,f;}           f3;
#       struct {int g,h; int i[10];} f4;
#       } u1,u2;
#int f(i) int i; {
#    switch(i) { /* What kind of structure to compare? */
        .text
        .align  1
L000:
        .globl  _.f
_.f:
        stm     r12,-52(r1)
        mr      r14,r0
        mr      r13,r1
        cal     r1,-52(r1)
        mr      r12,r2
        mr      r15,r12
        sis     r15,1
        cli     r15,4
        jh      L0C0
        a       r15,r15
        get     r2,$L02A
        a       r15,r2
        lhas    r15,0(r15)
        a       r15,r2
        br      r15
L02A:
        .short  L052-L02A
        .short  L034-L02A
        .short  L052-L02A
        .short  L074-L02A
        .short  L08C-L02A
#       case 1: return u1.f1.x == u2.f1.x &&
#                      u1.f1.y == u2.f1.y;
#       case 2: return u1.f2.c == u2.f2.c &&
L034:
        get     r2,$_u1
        ls      r3,8(r2)
        get     r4,$_u2
        ls      r5,8(r4)
        c       r3,r5
        jne     L0BE
        bx      L064
        ls      r3,0(r2)
#                      u1.f2.a == u2.f2.a &&
#                      u1.f2.b == u2.f2.b;
#       case 3: return u1.f3.e == u2.f3.e &&
L052:
        get     r2,$_u1
        ls      r3,0(r2)
        get     r4,$_u2
L064:
        ls      r5,0(r4)
        c       r3,r5
        jne     L0BE
        ls      r2,4(r2)
        ls      r3,4(r4)
        c       r2,r3
        jne     L0BE
        j       L0BA
#                      u1.f3.f == u2.f3.f;
#       case 4: return u1.f4.g == u2.f4.g &&
L074:
        get     r2,$_u1
        ls      r3,0(r2)
        get     r4,$_u2
        bx      L0A0
        ls      r5,0(r4)
#       memcmp(u1.f4.i,u2.f4.i,sizeof(u1.f4.i)) !=0;
#       case 5: return u1.f4.h == u2.f4.h &&
L08C:
        get     r2,$_u1
        ls      r3,4(r2)
        get     r4,$_u2
        ls      r5,4(r4)
L0A0:
        c       r3,r5
        jne     L0BE
        inc     r2,8
        cal     r3,8(r4)
        cal     r4,40(r0)
        balix   r15,_.memcmp
        l       r0,4(r14)
        cis     r2,0
        je      L0BE
L0BA:
        lis     r2,1
        j       L0C0
L0BE:
        lis     r2,0
L0C0:
        mr      r1,r13
        lm      r12,-52(r1)
        br      r15
        .long   0xDF07DFC8    # First gpr=r12
        .short  0x1D00        # npars=1, off=0
        .data   1
        .globl  _f
_f:
        .long   L000
        .long   _memcmp
        .align  2
        .data
```

In summary:

1. Cross-jumping is an amazingly effective optimization.
2. Toggle "`Optimize_xjmp`" is set `On` by default and turning it `Off` disables all cross-jumping.
3. Toggle "`Optimize_xjmp_space`" is `On` by default and turning it `Off` disables cross-jumping optimization that decreases space at the expense of time.

The cross-jumping optimization adds perhaps 20% to the execution time of the code generator phase of the compiler, thus perhaps 3% overall.

INDEX OF APPENDIX C

Starting below is a "permuted key word in context" index for this document. In the center column is the particular key word W being indexed, in the context of a phrase or sentence containing W. The phrase appears to the left and right of W.

Occasionally the text of the phrase preceding W does not fit in the space to the left of W. In that case the index entry looks like

is text that was too long to precede the WORD being indexed. This 7

where the first word "This" of the sentence did not fit on the left. Similarly the text to the right of W can be crowded:

right. This WORD is followed by too much text on the 7

where "the right" did not fit on the right.

After locating an entry, proceed directly to the referenced page.

ORDERING INFORMATION

Copies of the *High C* TM *Language Reference Manual* may be ordered directly from MetaWareTM. The manual retails for $16.95 and is available at an educational discounted price of $12.95.

If your system includes the Professional Pascal TM compiler, you may want the three-manual set, including the programmer's guide, primer, and language extensions manual. This set retails for $32.95 and is available at an educational discounted price of $24.95. The manual set may also be ordered from MetaWare.

These prices include mail/shipping costs. California residents please add 6.5% sales tax. Please send: (1) an indication of educational affiliation, if appropriate, and (2) a check, money order, or written authorization to charge to your MasterCard or VISA account, with account number and expiration date to:

MetaWare Incorporated
903 Pacific Avenue, Suite 201
Santa Cruz, CA 95060-4429
(408) 429-META (=6382)

This page intentionally left blank.

Academic Information Systems 4.2
for the IBM RT PC
READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

If you wish, give your name, university or site, mailing address, and date:

____

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments, or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Academic Information Systems
University Support, Dept. 6FR
P.O. Box 10500
Palo Alto, CA 94303-9974

Fold and tape | Please Do Not Staple | Fold and tape

IBM®