R1000 Command Interfaces

1. CLI

1.1. Overview

The Command Line Interpreter (CLI) interface is the lowest level interface activated on an R1000. It is available via several means:

- Initial Machine Power ON
- After a system shutdown (via BREAK or Operator.Shutdown)
- After a System Crash

1.2. Commands

This section describes the commands available from the CLI level. Generally, at the CLI command level, commands are programs which the CLI executes using the form:

```
CLI> x Command
```

where *Command* is one of the following:

```
Bootinfo        Cedit       Checkdisk
Commx           Configure   Crashdump       Crashload
Diskmd          Diskx       Display         Erasedisk
Expmon          Findseg     Gc              Initioa
Initstate       Iox         Loadee          Loader
Log             Look        Memmacs         Mt
Novram          Rdiag       Rdm             Recovery
Sam             Scan        Slew            Starter
Stat            Tapex       Trace           Update_Eeprom
```

The following are Features/limitations:

- Filenames are limited to 30 characters.
- All commands can be abbreviated.
- The following special characters are recognized:

Character	Description
*	Wildcard used in filename, match any character(s)
^P	PUP command, Power Logon
^S	
^Q	
^H	
^C	
^R	
^U	

Figure 1.1 - CLI Special Characters

1.2.1. BOOTINFO

The BOOTINFO program is used to determine the software configuration used during the last successful system boot. It displays information about the microcode, as well as the ten subsystems which are loaded from the DFS. These subsystems are:

```
ADA_BASE
MACHINE_INTERFACE
KERNEL_DEBUGGER_IO
KERNEL_DEBUGGER
KERNEL
ENVIRONMENT_DEBUGGER
ABSTRACT_TYPES
MISCELLANEOUS
OS_UTILITIES
ELABORATOR_DATABASE
```

The information supplied for microcode is the DEC System 20 pathname of the MOM and DELTA as well as the date on which the control store image was bound. The information supplied for each subsystem is simply the name of the .MLOAD file which was used to load the subsystem into the R1000. By convention the name of the .MLOAD file provides sufficient information to find the ADA sources involved.

1.2.2. CEDIT

The CEDIT program is used to edit R1000 configurations. These configurations specify a group of microcode and software subsystems which may be loaded into the R1000 processor. In addition a configuration also specifies certain attributes of the systems hardware. Usually configurations are distributed with system software releases and should not be edited. The CEDIT program is intended for use by Rational Support Personnel when debugging or working around certain problems with new releases. Changing the contents of a configuration may make it difficult or impossible for Rational to determine which versions of software comprise the running system.

The Rational Environment is made up of several dozen subsystems. Most of these subsystems reside within the Environment's virtual memory system. Ten of the subsystems are loaded from the DFS during system boot and comprise the portions of the Environment which are required for the virtual memory system to function and for the remainder of the subsystems to be located and elaborated. These subsystems are:

```
ADA_BASE
MACHINE_INTERFACE
KERNEL_DEBUGGER_IO
KERNEL_DEBUGGER
KERNEL
ENVIRONMENT_DEBUGGER
ABSTRACT_TYPES
MISCELLANEOUS
OS_UTILITIES
ELABORATOR_DATABASE
```

For each of these subsystems a configuration file contains information about where the subsystem is within the DFS. A subsystem may be located in one of three ways:

- User is queried at load time for the subsystem. This method is only used internally for Environment development.
- The subsystem is explicitly named within this configuration. This is the normal method of locating subsystems.
- The subsystem is explicitly named in the STANDARD configuration and its location should be determined from the STANDARD. This method is useful for defining deltas from the STANDARD.

To invoke the configuration editor type:

```
CLI> x cedit
```

The program will first begin to edit the system's hardware configuration as follows:

```
Change hardware configuration [N] ?
```

If you want to modify the hardware configuration enter "Y", otherwise enter the default.

Editing Hardware Configuration

If you have chosen the default, the editor will proceed to the software configuration. If you request to modify the hardware configuration it will proceed as follows:

If the system is a Series 100 you will be asked:

```
Is this a multi-processor ? N
```

This attribute is only used on Rational's internal Series 100 processors which have been modified for multi-processing.

```
Does this processor have 8 MB memory boards ? Y
```

All Rational systems currently use eight megabyte memory modules. This question should be answered "Y".

For each of the four possible memory modules you will be asked:

```
Does memory board <n> exist ? Y
```

All Rational systems currently use all four possible memory modules. All of these four questions should be answered "Y". It is possible to run with only 3 memory modules, but not less.

Editing the hardware configuration is now done.

Editing Software Configuration

Editing the software configuration is really done in two parts. First some information about how the booting process works is needed followed by information about each of the ten subsystems. Before that the name of the configuration you are creating or modifying is needed. The CEDIT program will ask:

```
Enter name of configuration to edit [STANDARD] :
```

Enter the name of the configuration on which you want to base your changes. This is not neccessarily the configuration you will be changing. You will then be asked:

```
Enter name of configuration to save [Configuration] :
```

where *Configuration* is the default configuration name to edit.

By default the editor will save your changes in the configuration you are basing them on. You may specify a different name if you wish. If you specify a different configuration the original will not be changed.

Now the editor will inquire about some booting options. These are:

```
Allow operator to enter CLI immediately ?
```

If this question is answered "Y" the operator will be asked if he/she wishes to enter the CLI at the beginning of the booting process.

```
Allow editing of configuration ?
```

If this question is answered "Y" the operator will be asked if he/she wishes to edit the configuration being used to boot the processor during the booting process.

```
Allow operator to enter CLI prior to starting the cluster ?
```

If this question is answered "Y" the operator will be asked if he/she wishes to enter the CLI between the time that microcode is loaded and macro-code is loaded.

```
Load kernel layer subsystems only ?
```

If this question is answered "Y" only the first five subsystems will be loaded before the R1000 processor is started. This feature is used to run certain R1000 diagnostics and exercisers.

Now the editor will ask some questions about the microcode and subsystems to be used to boot the processor. These questions will be asked eleven times, once for each of the following:

```
MICROCODE
ADA_BASE
MACHINE_INTERFACE
KERNEL_DEBUGGER_IO
KERNEL_DEBUGGER
KERNEL
ENVIRONMENT_DEBUGGER
ABSTRACT_TYPES
MISCELLANEOUS
OS_UTILITIES
ELABORATOR_DATABASE
```

The questions are:

```
Take <subsystem-name> from STANDARD ?
```

This question will not be asked if you are editing the STANDARD configuration. If you answer "Y" to this question you will not be asked any more questions about this subsystem.

```
Should this configuration query for <subsystem-name> ?
```

If you answer this question "Y" then at boot time the macro-state loader will query about where the subsystem is located.

```
Enter name for <subsystem-name> ?
```

The answer to this question tells the macro-state loader where to get code for this subsystem. It is the name of an MLOAD file. If you answered "Y" to the query question the name you enter here will be the default answer when the operator is queried at boot time.

The configuration editor will now save the configurations modified during the editing process. If this succeeds it will display the message:

```
Configuration saved!
```

In any case the CEDIT program will terminate.

1.2.3. CHECKDISK

The CHECKDISK program performs a non-destructive, fast, read-only surface analysis of a formatted, labeled disk. CHECKDISK will avoid previously detected bad blocks on a labeled disk, reporting only errors which are not known.

Run the CHECKDISK program by typing:

```
CLI> x checkdisk
```

The CHECKDISK program will ask for the disk unit number to check. The disk must have been previously formatted and labeled with the RECOVERY program. Next the program will ask for the number of passes desired. Normally a single pass is sufficient but for internal or testing purposes several passes may be desired. The program will then ask if you want error information displayed for all defects or only previously-undetected defects. Normally only previously undetected error information is desired but at times all information is useful.

Once the program begins it will display the current cylinder number constantly and print a pass counter at the conclusion of each pass. The time required for a pass is dependent on the disk but may be calculated as follows:

```
((C*H*S)/16)/R seconds

where:
C = number of cylinders
H = number of heads
S = number of sectors
R = spindle revolutions per minute

Fujitsu 2351 (EAGLE) example:
((842*20*48)/16)/3961 = 12.75 minutes
```

1.2.4. CLI

The CLI is the I/O Processor's Command Line Interpreter. It is used to invoke programs, create files, delete files, display files, list files, set the time and display the time. The CLI makes use of the DFS macro user interface with which you should be familiar.

The CLI accepts the following commands:

Command	Description
COPY	Copies data from one file into another file.
CREATE	Creates a file.
DELETE	Deletes a file.
DIRECTORY	List files matching a given file specification.
LOCAL	Returns operations to the local console (issued from a remote connection).
PRINTER	Enables/disables hardcopy of console output.
REMOTE	Transfers operations to the diagnostic modem.
RENAME	Changes the name of a file.
TIME	Display/change the time.
TYPE	Displays the contents of a text file.
X	Executes a program.

Figure 1.2 - CLI Imbedded Commands

1.2.4.1. COPY

The COPY command creates a new file and copies the contents of an existing file into the new file. Command syntax is:

```
CLI> copy Existing_File New_File
```

The new file must **not** already exist. If any disk errors are encountered the command is aborted but the output file *New_File* is not deleted, and its contents are indeterminant.

Switch	Description
/D	Delete the output file if it already exists.

Figure 1.3 - COPY Switches

Example:

```
CLI> copy/d Existing_File New_File
```

where *Existing_File* and *New_File* are valid filenames.

1.2.4.2. CREATE

The CREATE command creates a file. Command syntax is:

```
CLI> create New_File
```

The new file must not already exist.

Switch	Description
/D	If a file by the given name exists, Delete the existing file before creating the *New_File*.
/SIZE=N	CREATE a file of N pages. Default = 1.
/CONTIGUOUS	Create the file with CONTIGUOUS disk space.
/I	Allow the user to enter text into the file.

Figure 1.4 - CREATE Switches

Example:

```
CLI> create/d/size=4/contiguous/i New_File
)This data will be placed into the file New_File.
))
```

The final character typed is a single ")" character on a line by itself. This terminates input and closes the file.

1.2.4.3. DELETE

The DELETE command deletes all existing files matching a given file specification. Command syntax is:

```
CLI> delete filename
```

Switch	Description
/V	Display the name of each file as it is deleted.
/C	Confirm that each file is to be deleted by asking the user.

Figure 1.5 - DELETE Switches

1.2.4.4. DIRECTORY

The DIRECTORY command is used to display information about all files matching a given file specification within the diagnostic file system. Command syntax is:

```
CLI> directory Filename
```

The size and creation time of each file matching the *Filename* is displayed along with a summary of the total number of files and total number of disk pages.

Switch	Description
/FULL	Causes information from the file's File Control Block to be displayed. This includes file attributes, allocation, and disk address information.

Figure 1.6 - DIRECTORY Switches

1.2.4.5. LOCAL

The LOCAL command is issued to a remote R1000 when customer support wishes to return control of that system to the on-site personnel.

```
CLI> local
```

See REMOTE.

1.2.4.6. PRINTER

The PRINTER command is used to enable or disable hardcopy output of all operations console transactions. Command syntax is:

```
CLI> printer on Line_Number
CLI> printer off
```

Line_number is the R1000 port number to which a line printer is attached. The printer should be on-line and should not use any software flow-control protocol. Hardware flow-control is permitted.

1.2.4.7. REMOTE

The REMOTE command is used by local personnel to transfer operations of an R1000 to Rational's Customer Support Response Center.

See LOCAL.

1.2.4.8. RENAME

The RENAME command is used to change the name of an existing file. Command syntax is:

```
CLI> rename Old_filename  New_Filename
```

Old_Filename must exist, *New_Filename* must not.

Switch	Description
/D	Delete the *New_Filename* if it already exists

Figure 1.7 - RENAME Switches

1.2.4.9. TIME

The TIME command may be used to display or set the time. It has immediate effects on the system's battery-backed-up clock/calendar. Command syntax is:

```
CLI> time                                  -- Display Time
CLI> time hh:mm:ss dd-mon-yy  -- Set the Time
```

where

HH	Hours since midnight in military (24 hour) format
MM	Minutes
SS	Seconds
DD	Day of the month
MON	One of JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
YY	Last two digits of the year

The DFS displays the time to within only 2 second granularity. The time command will set the time to the exact second.

1.2.4.10. TYPE

The TYPE command may be used to display the contents of all files matching a given file specification. Command syntax is:

```
CLI> type Filename
```

Switch	Description
/V	Display the name of each file prior to its contents

Figure 1.8 - TYPE Switches

1.2.4.11. X

Executes a program (Command) as defined in this chapter.

```
CLI> x Program_Name
```

where *Program_Name* is something like BOOTINFO, etc., as documented in this chapter.

1.2.5. COMMX

The COMMX program is a DH-11 communications multiplexer exerciser. It can be used to exercise either input or output to the R1000's RS-232C lines. To invoke the program type:

```
CLI> x commx
```

The program will query for INPUT or OUTPUT testing. Once you have answered it, proceed as follows:

```
INPUT =>
```

Any character coming into the DH-11 will be recognized and a message such as:

```
Line N on unit M received ascii XX
```

will be sent to the console as well as out the line from which the character was received. Typing any character on the console will terminate the test.

```
OUTPUT =>
```

A character string like:

```
This is line N on unit M
```

will be sent to all DH-11 lines. Typing any character on the console will terminate the test.

1.2.6. CONFIGURE

The CONFIGURE program is invoked during system boot and diagnosis to power the R1000 processor (Series 100 only), reset the R1000 processor, and verify the presence of all R1000 processor boards. It may not be invoked by the user.

1.2.7. CRASHDUMP

The CRASHDUMP program is used to capture all R1000 processor state on a magnetic tape for use in debugging system software and microcode problems. After some types of system crashes the operator will be asked if he/she wishes to take a CRASHDUMP. In most cases the operator should. This program is fairly self-explanatory but requires some input from the system operator. At times when the CRASHDUMP program is not invoked automatically the operator may invoke it as follows:

```
CLI> x crashdump
```

The CRASHDUMP program queries the operator as follows:

```
Tape unit number 0 .. 3 [0] ?
Tape density is PE(GCR), you can manually change it now.
Volume Id, (1..6 characters) ?
Please enter any comments or information that may help isolate or reproduce
this problem.  To terminate the comment, enter a ")" on a line by itself.
```

Answer these questions in the obvious fashion. The Volume ID field is used to track crash dump tapes. You should keep a log of the Volume ID you enter. The comments are of great importance when debugging from a CRASHDUMP tape. They should include processor location, your name, your phone number, and any information about what seemed to cause the system to crash, what users were on the system, and what kind of jobs were they running.

The CRASHDUMP program takes a fair amount of time to write the entire tape. The tape should be at least 1200 feet long for a GCR CRASHDUMP. The program displays the following information while running:

```
Saving state for board JKLMFQTVS [OK]
Saving Special Registers [OK]
Saving Trace Rams [OK]
Dumping CRASH_DUMP.COMMENTS [OK]
Dumping IOP_DUMP1 [OK]
Dumping IOP_DUMP2 [OK]
Dumping CRASH_DUMP.SAVED_STATE [OK]
Dumping CRASH_DUMP.REGISTERS [OK]
Dumping CRASH_DUMP.TAG_STORE [OK]
Dumping CRASH_DUMP.MEMORY 0123456789ABCDEF [OK]
Crash Dump is complete.
```

Once the CRASHDUMP is finished, remove the tape from the R1000, write-protect it, and forward it to the Rational Customer Support Response Center. Be sure to place a legible label on the tape with your name and phone number, and something to prevent the tape leader from unravelling/wrinkling during shipment.

1.2.8. CRASHLOAD

The CRASHLOAD program loads dump tapes produced by the CRASHDUMP program into an R1000 for debugging purposes. This program is intended for internal use only.

1.2.9. DISKMD

The DISKMD program is a DFS-based disk utility which is useful for a wide range of disk problems. It is USER_INTERFACE based and has several commands which act on one of two memory buffers. One buffer is the read buffer, the other is the write buffer. Each buffer is 1024 bytes in size, the same as an R1000 disk block.

Command	Description
COPY	The COPY command copies the contents of the read buffer into the write buffer.
CTS	The CTS command requires a single decimal argument (the block number) which is converted into cylinder, track, and sector and then inserted into the macro evaluation buffer.
DATA	The DATA command requires a single hex argument which is truncated to 16 bits and used to fill the write buffer.
DBN	The DBN command requires three decimal arguments (Cylinder, Track, Sector) which are converted into a disk block number and inserted into then macro evaluation buffer.
DISPLAY	The DISPLAY command displays the contents of the read buffer.
EDIT	The EDIT command requires two hex arguments. The first is used as an address in the write buffer and the second is a 16-bit data word which is inserted into the write buffer at the address specified.
RD	The RD command requires three decimal arguments which are the cylinder, track, and sector of a 1KB disk block to be read from the current unit into the read buffer. If any errors are encountered during the read, the buffer will only contain data from sectors successfully read. Other portions of the read buffer will remain unchanged.
RECAL	The RECAL command recalibrates the current disk unit.
SEEK	The SEEK command requires a single decimal argument which is interpreted as a cylinder number. The heads of the currently-selected drive are positioned to that cylinder but no transfer is done.
STATUS	The STATUS command displays the disk status from the last error for the current unit. If no errors have occurred the displayed status is indeterminate.
VERIFY	The VERIFY command compares the contents of the read buffer to the contents of the write buffer. Discrepancies are displayed on the console.
WR	The WR command requires three decimal arguments: a cylinder, track, and sector number. A 1KB transfer from the write buffer to that sector is started on the current unit. If any errors occur the transfer is aborted.
WRENABLE	The WRENABLE command is issued to write-enable the current unit. It remains in effect until the UNIT command is issued.
UNIT	The UNIT command requires a single decimal argument which is used for all disk-specific I/O from then on. The UNIT command software write protects the newly selected disk until the WRENABLE command is issued.

Figure 1.9 - DISKMD Commands

1.2.10. DISKX

The DISKX program is a DFS-based disk exerciser. It is intended to be used for quick disk subsystem integrity testing as well as long-term reliability testing. To invoke the program type:

```
CLI> x diskx
```

The program will size the system and query the operator about testing each disk unit. After these questions the exerciser will begin testing. Once testing has begun the exerciser may be stopped by typing control-G. Any other character will cause the program to display status information about each drive. This status information includes the total number of bytes transfered, total hard errors, total soft errors, and current head location.

The exerciser does transfers in the following manner:

1. Writes random data to a random block in the diagnostic region.
2. Checks the data just written.
3. Reads a random block from anywhere on the disk.

These three steps are repeated for each unit with all units running in parallel until the test is stopped.

1.2.11. ERASEDISK

The ERASEDISK program is used to comply with certain national security guidelines when removing disks from a secure site. This program will format the desired disk unit three times, using data supplied by the operator as a format pattern. The program is self explanatory. To invoke the ERASEDISK program type:

```
CLI> x erasedisk
```

The ERASEDISK program will overwrite every disk block on the disk. No information will be retained. Once the ERASEDISK program has erased a disk you will need a defects tape to re-instate the bad block region and re-format the disk. When erasing disks, it is always best to erase unit 0 last since this command is generally located on that unit.

1.2.12. EXPMON

EXPMON is the DFS-based experiment monitor. It is the basis of all low level hardware, software, and microcode debugging. The experiment monitor makes use of the macro user interface and nearly all interaction with the program is based on a collection of nearly 1000 macros. The basic command set, however, is fairly small. To use EXPMON you should be familiar with R1000 hardware, R1000 micro-architecture, R1000 macro-architecture, and the EXPMON macros.

Command	Description
XEQ	The XEQ commands causes an experiment to get loaded from the disk, parameterized, downloaded to a given board, executed, and uploaded. Once the experiment has been uploaded any output parameters are placed in the evaluation buffer. Syntax: `XEQ board-name experiment-name [parameter1,parameter2,...]`
POLL	The POLL command polls a given board and inserts the board's status into the evaluation buffer. Syntax: `POLL` *`Board_Name`*
RESET	The RESET command sends a reset instruction to a given board. This causes the boards DFSM to be reset. Syntax: `RESET` *`Board_Name`*
CONTINUE	The CONTINUE command causes an experiment running on a given board to continue from the paused state. Syntax: `CONTINUE` *`Board_Name`*
MEMn_EXISTS	Each of the MEMn_EXISTS (where n is 0..3) commands places a boolean into the evaluation buffer corresponding to the existence of that memory board.
QUAD_DENSITY	The QUAD_DENSITY command places a boolean into the evaluation buffer which is false if the machine has 2MB memory boards and is true if the machine has 8MB memory boards.

Figure 1.10 - EXPMON Commands

Board_Names recognized by the experiment monitor are as follows:

```
IOA     I/O adaptor  (Series 100 only)
SYS     Sysbus board (Series 100 only)
IOC     I/O channel  (Series 200 only)
TYP     Type ALU
VAL     Value ALU
SEQ     Micro-sequencer
FIU     Field isolation unit
MEM0    Memory board 0 (right-most)
MEM1    Memory board 1
MEM2    Memory board 2
MEM2    Memory board 3 (left-most)
ALL     All R1000 processor boards excluding the I/O adaptor.
```

1.2.13. FINDSEG

The FINDSEG program is used internally to determine in which subsystem a given code segment name is located. The program is invoked by typing:

```
CLI> x findseg <some code segment name>
```

If the code segment name is left off of the command line the program will prompt for one when running.

If the given segment name was loaded by the LOADER during the last system boot then FINDSEG will display the subsystem name in which it is located.

1.2.14. GC

The GC is a disk garbage collector which can delete unnecessary files within the DFS. The garbage colletor scans all existing MLOAD files and retains a list of all SEG files referenced. The program then scans all SEG files in the DFS-filesystem, deleting the ones not referenced by any MLOAD files. The garbage collector will recover disk space if unnecessary MLOAD files are deleted manually.

1.2.15. INITIOA

The INITIOA program allows the operator to change two parameters which are stored in NOVRAM on the IOA (Series 100) or the IOC (Series 200). These parameters are the *Cluster Id* and the *Phone Number* of the Rational Customer Support Response Center which is used for remote access.

Cluster Id

The system's *Cluster Id* is a unique, 6 digit, installation identifier. It is the key used to access information about the system in the Customer Support Database.

Phone Number

The PHONE_NUMBER is used by the system to begin remote diagnosis of R1000 detected problems. It is not used to make voice contact with the Rational Response Center.

The INITIOA program is intended for use during system installation by Rational Support Personel. To invoke the INITIOA program type:

```
CLI> x initioa
```

The program will repond by asking two questions:

```
Enter CLUSTER ID [<current-id>] : Cluster ID

Enter phone number to be used for remote diagnostics.
Include all PBX codes required to gain outside access.
The following characters may be imbedded at any point :
    W => Wait for dial tone.
    D => Pause 3 seconds.
    P => Subsequent digits are pulse dialed
    T => Subsequent digits are tone dialed (DTMF).
    - => No effect, used for clarity only.

Enter phone number [<current-phone-number>] : Phone Number
```

If the <current-id> is not correct, enter the correct *Cluster Id*. If the <current-id> is correct simply type <cr> to retain it.

The phone number is a character string which contains embedded characters to inform the modem about the installation site's phone system.

The following example is used to illustrate how INITIOA is used.

1. The actual phone number is 800-555-1212.
2. A PBX is in use which requires entering a "9" to gain outside access.
3. The site uses a third party phone network like "SPRINT" or "MCI" which is accessed by dialing 800-123-4567. The site's access code to this network is

121212.

If a person were making this phone call he/she would do the following:

1. Wait for the PBX dial tone
2. Dial "9".
3. Wait for the local phone systems dial tone.
4. Dial 800-123-4567 to gain access to the 3rd party network.
5. Wait for a tone.
6. Enter the access code "121212"
7. Wait for a dial tone.
8. Dial 800-555-1212 to finally contact Rational.

The modem can be instructed to do nearly the same thing. The only difference would be step 5. The modem cannot wait for a tone but instead it can be instructed to wait until it is certain that the tone has happened (by inserting 3 second delays). The modem's instructions might look like this:

```
W9W8001234567DDD121212W8005551212
```

To make this look more legible we can insert "-" characters. These are ignored by the modem. A more legible set of instructions might be:

```
W-9-W-800-123-4567-DDD-121212-W-800-555-1212
```

Once both questions have been answered the INITIOA program will store the new data and terminate.

1.2.16. IOX

IOX is a DFS based I/O exerciser. It is a combination of the DFS based tape exerciser, TAPEX, and the DFS based disk exerciser, DISKX. The IOX program may only be run on Series 200 processors due to memory constraints. To invoke the IOX program type:

```
CLI> x iox
```

The exerciser will prompt you as follows:

```
Simulate packet requests [Y] ?
```

Answering "Y" to this question causes IOX to use the same entry points to the I/O Processor Kernel as the R1000 would use during system operation. It provides for more extensive testing of system integrity, allows more parallelism, and increases I/O throughput.

```
Exercise Tapes [Y] ?
```

Answering "Y" to this question will cause IOX to ask if you want to exercise each tape drive which is in the system and is "ready". A tape drive is "ready" if it has a tape mounted, loaded, and write- enabled. As many as twelve tape drives may be exercised. Tape testing will destroy all data written to the tape.

```
Exercise Disks [Y] ?
```

Answering "Y" to this question will cause IOX to ask if you want to exercise each on-line disk drive. As many as sixteen disk drives may be exercised. Disk testing will only write to the diagnostic area of a disk. This should cause non-destructive results. If a disk unit is in question a full system backup should be available as certain disk malfunctions may result in writes to user data.

If you have requested that disks be exercised you will be asked:

```
Do all I/O to the same cylinder [N] ?
```

Answering "Y" to this question will cause more transfers because the disk heads will never be moved. Answering "N" will provide for testing more comparable to system usage because seeks will be quite frequent.

Once the exerciser has asked all its questions it will begin to test the devices selected. It tests the tapes and disks in the same fashion as DISKX or TAPEX. Refer to the documentation for those programs for details of testing. While IOX is running you may type **^G** (Control-G) to terminate it. Typing any other character will cause status displays which are similar to those produced by TAPEX and DISKX. Refer to those sections for details.

1.2.17. LOADEE

The LOADEE program is used to re-program the R1000 system's bootstrap Electrically-Erasable Programmable Read-Only Memories (EEPROMs). These devices contain executable data used to bootstrap the system. The EEPROMs are located on the IOA (Series 100) and on the IOC (Series 200). The LOADEE program is intended to be used by Rational Support Personnel when installing new system software releases. Documentation for the release being installed is required to successfully run the LOADEE program.

Prior to re-programming the bootstrap EEPROMs you must have loaded the desired program images into the system's DFS. To invoke the program type:

```
CLI> x loadee
```

The program will query as follows:

```
Enter I/O Processor Bootstrap file name :
Enter I/O Adaptor Cluster Manager file name :
```

Respond to these questions as instructed in the software installation instructions. The LOADEE program will transfer the contents of the specified file into the system's EEPROMs and then re-boot the R1000.

1.2.18. LOADER

The LOADER program is used to boot the R1000. Depending on the BOOT/CRASH/MAINTENANCE options the LOADER will either prompt for a configuration to boot or simply use the STANDARD configuration.

The LOADER will first initialize the processor by invoking various experiments. Next, the microcode loader for the machine (SBUSULOAD for the Series 100 and DBUSULOAD for the Series 200) is initialized. The microcode loader will reload the control store, register file, and dispatch RAMs as needed from the microcode image specifed by the configuration file.

The LOADER will proceed to start the R1000 by invoking experiments. At this stage the R1000 will begin executing microcode to further initialize the processor and eventually become idle.

The LOADER will begin to load subsystems into the R1000 by extracting the names of MLOAD files from the configuration specified and interpreting the directions contained in them. Once all of the subsystems have been loaded into R1000 memory the LOADER will calculate the transitive closure of the subsystems, build instructions for the kernel elaborator and send those to the R1000 as well. Finally the LOADER will send a start message to the R1000. The R1000 will place a constant task name on the run queue and thus begin elaborating the environment.

Once done the LOADER invokes the monitor program which performs various tasks while the R1000 is running.

1.2.19. LOG

The LOG program allows the user to examine or initialize the DFS-based error log. This error log contains information about device errors during DFS I/O, system crash history, diagnostic failures, etc. The LOG program is interactive and is invoked by typing:

```
CLI> x log
```

Once the LOG program has been invoked it will display information about the contents of the error log. The error log is a ring buffer of 2048 entries. Newer information overwrites older information in the error log. The LOG program allows selective examination of log entries (displayed starting with the most recent first). Once you have finished examining the log you will be asked:

```
Update log header [Y] ?
```

Answering "Y" to this question will cause all entries in the error log to be marked as old. This allows subsequent examinations of the error log to filter out data already seen. Entries marked as old can still be examined until they are overwritten within the ring buffer.

To initialize the error log invoke the LOG program as follows:

```
CLI> x log initialize
```

The LOG program will ask if you really want to initialize the log. This will result in all log data being discarded. The LOG should be initialized when a new DFS is built on a system.

1.2.20. LOOK

The LOOK program can be used to examine a file within the DFS. It is invoked by typing:

```
CLI> x look Filename
```

The program will prompt for a file to examine if one is not specified on the command line. Once the file has been opened the LOOK program prompts for user input with a ">". You may now type either one or two octal arguments. The first argument is assumed to be an offset of 8-bit bytes into the file. The second word is assumed to be a count of 16-bit words to be displayed. If the second argument is not provided it is assumed to be 10 (8 decimal). The LOOK program takes the address and sets the low eight bits to zero. It then takes the count and rounds it up to the nearest multiple of eight. LOOK then displays the portion of the file specified by the modified address and count as 16-bit octal words. To terminate the program type "BYE" to the prompt. LOOK is intended for internal use only.

1.2.21. MEMMACS

The MEMMACS program provides a performance accelerator for several experiment monitor macros. It provides the following functions:

```
Logical memory read
Physical memory read
Logical memory write
Physical memory write
Tagstore display
Physical tag read
Physical tag write
Memory board state display
RDR display
TVR display
Compute ECC
```

This program should not be invoked except with the experiment monitor macros for which it is intended.

1.2.22. MT

The MT program is a DFS-based magnetic tape utility. It is used to transfer DFS files between a DFS disk and an "MT" format tape. The MT program utilizes the macro user interface. It is invoked by typing:

```
CLI> x mt
```

The following are commands executed by the MT interface:

```
Load        Dump        Rewind      Unload
```

1.2.22.1. LOAD

The LOAD command is used to transfer files from a tape to a DFS disk. The LOAD command requires no arguments and will transfer all files on the tape. The LOAD command leaves the tape positioned at End of Tape (EOT).

Switch	Description
/N	The /N switch causes the LOAD command to read the files from tape but not write them to disk. This switch may be used in conjunction with the /V switch to see what's on a tape or to position the tape at EOT to enable appending files to an existing tape.
/V	The /V switch causes the MT program to display the name of each file read from tape.
/UNLOAD	The /UNLOAD switch causes the LOAD command to unload the tape rather than leaving the tape positioned at EOT.
/UNIT=	The /UNIT= switch may be used to specify a specific tape unit for the transfer. If this switch is not present, unit zero will be used.

Figure 1.11 - MT LOAD Switches

Examples:

```
MT> load/v/unload/unit=1
```

To append files to an existing tape type:

```
MT> load/n
MT> dump file1,file2,...
```

The LOAD/N command will position the tape at EOT but not transfer any data to the disk. The dump command will then begin to transfer the specified files to the end of the tape.

1.2.22.2. DUMP

The DUMP command is used to transfer files from a DFS disk to a tape. The DUMP command will leave the tape positioned at EOT upon transfer of the last file. The dump command requires at least one argument. This argument is the file specification to dump to tape. More than one file specification may be present on the command line. They will be processed in the order they appear.

Switch	Description
/GCR	The /GCR switch will force the file to be written in GCR format if the tape drive supports remote density selection. The /GCR switch may only be used if the tape is at BOT. GCR format records at 6250 bits per inch.
/PE	The /PE switch will force the file to be written in PE format if the tape drive supports remote density selection. The /PE switch may only be used if the tape is at BOT. PE format records at 1600 bits per inch.
/LONG_GAPS	The /LONG_GAPS switch will force the tape to be written with long or variable inter-record gaps if the tape drive supports remote gap selection. This will increase tape throughput.
/SHORT_GAPS	The /SHORT_GAPS switch will force the tape to be written with short inter-record gaps if the tape drive supports remote gap selection. This will increase tape capacity.
/THRESHOLD=	The /THRESHOLD switch helps the MT program optimize throughput. It requires a single argument. If a file being DUMPed is larger than the THRESHOLD that file will be dumped in high-speed mode. If the file is smaller it will be dumped in low speed mode. The MT program uses reasonable defaults but the /THRESHOLD allows the user to optimize further.
/UNIT=	The /UNIT= switch may be used to specify a specific tape unit for the transfer. If the /UNIT= switch is not present, unit zero will be used.
/UNLOAD	The /UNLOAD switch causes the LOAD command to unload the tape rather than leaving the tape positioned at EOT.
/V	The /V switch causes the MT program to display the name of each file read from tape.

Figure 1.12 - MT DUMP Switches

Examples:

```
MT> dump/v/gcr/threshold=25/short_gaps file1,file2,*file3*
```

To append files to an existing tape type:

```
MT> load/n
MT> dump file1,file2
```

The LOAD/N command will position the tape at EOT but not transfer any data to the disk. The dump command will then begin to transfer the specified files to the end of the tape.

1.2.22.3. REWIND

The REWIND command will reposition the tape to the Beginning of Tape (BOT).

Switch	Description
/UNIT=	The user may select a unit with the /UNIT= switch. By default unit zero is rewound.

Figure 1.13 - MT REWIND Switches

Examples:

```
MT> rewind/unit=2
```

1.2.22.4. UNLOAD

The UNLOAD command will reposition the tape to BOT and then unload the tape from the drive.

Switch	Description
/UNIT=	The user may select a unit with the /UNIT= switch. By default unit zero is unloaded.

Figure 1.14 - MT UNLOAD Switches

Examples:

```
MT> unload/unit=3
```

1.2.23. NOVRAM

The NOVRAM program is used to modify and display the NOn-Volatile RAMs located on each R1000 processor board. The NOVRAM program is used during the boot process to insure that the NOVRAMs contain valid information. The NOVRAM program may also be invoked by typing:

```
CLI> x novram
```

The program will load the contents of all boards' NOVRAMs and then display a menu. The current menu options are:

```
0 => exit
1 => modify
2 => display
```

The exit option causes the program to terminate. The modify option allows the user to modify the NOVRAM of a given board. Default answers are provided to all questions during the modification process. The default values are either the current values if the NOVRAMs checksum is correct, or best guess values if the boards NOVRAM does not checksum successfully. The display option provides a table of information about each board in the R1000 processor. Included are the board part number, serial number, artwork revision level, ECO level, and manufacturing date.

1.2.24. RDIAG

The RDIAG program is used to run FRU diagnostics. It is an interactive program based upon the macro user interface. To run RDIAG, type:

```
CLI> x rdiag
RDIAG>
```

The following commands can be executed from the RDIAG interface:

```
TEST     RUN      ERRMESS      INIT_STATE       ISOLATE
TRACE    ULOAD    MARGIN
```

and are explained in more detail on the following pages.

1.2.24.1. TEST

The TEST command is used to run all diagnostics pertaining to a given FRU.

Format:

```
DIAG> test FRU
```

Switch	Description
/1	Execute only PHASE I tests for this FRU.
/2	Execute only PHASE I and PHASE II tests for this FRU.
/3	Execute PHASE I, II, and III tests for this FRU.

Figure 1.15 - DIAG TEST Switches

1.2.24.2. RUN

The RUN command is used to execute a single diagnostic program.

Format:

```
DIAG> run [FRU] Diagnostic
```

The fru argument is only allowed when invoking diagnostics which test multiple FRUs. These include P2UCODE, P3UCODE, and all memory FRUs.

1.2.24.3. ERRMESS

The ERRMESS command is used to extract an error message from an error message file. This command is primarily intended for debugging purposes.

Format:

```
DIAG> errmess <error-message-file>,<error-number>,[<error-string>]
```

1.2.24.4. INIT_STATE

The INIT_STATE command is used to restore the R1000 processor to a known, benign, state.

1.2.24.5. ISOLATE

The ISOLATE command causes FRU diagnostics invoked with the TEST or RUN commands to invoke other diagnostics to isolate a problem.

Format:

```
DIAG> isolate {On | Off}
```

1.2.24.6. TRACE

The TRACE command enables or disables the single line of output associated with the execution of an experiment. This is most useful when debugging with a remote connection at a low baud rate.

Format:

```
DIAG> trace {On | Off}
```

1.2.24.7. ULOAD

The ULOAD command controls the loading of control-store for certain FRU diagnostics.

Format:

```
DIAG> uload {on | off}
```

1.2.24.8. MARGIN

The MARGIN command is used to margin the power and clocks utilized by the R1000 processor.

Format:

```
DIAG> margin {power | clock} {high | normal | low}
```

1.2.25. RDM Does Not Apply to 400

The RDM program is used to recover disk-defect information from new disks which have not yet been formatted. Some disk manufacturers place information about disk defects on the disk itself to allow the disk formating process to be as automatic as possible. The RDM program uses features of the SL-121 controller to retrieve this information. To run the RDM program either boot it from tape or invoke it by typing:

```
CLI> x rdm
```

The RMD program will ask several questions as follows:

```
Enter unit number of virgin disk :
```

Type the disk unit number as set on the disk drive itself.

```
Enter HDA serial number as shown on HDA :
```

Enter the disk's HDA serial number. Note that this is different from the disk drive's serial number as defects are tracked by HDA, not drive.

```
Enter the number of bytes-per-sector as jumpered at the disk :
```

This information is required to calculate the sector number in which a given defect is located. For the disks which Rational currently uses the answer to this question is:

```
Fujitsu 2351 (EAGLE)     585
Fujitsu 2361 (XP)        620
```

The answer to this question is vital. Neither the program nor the controller can determine if the information you have provided is correct. An incorrect answer will result in incorrect mapping of disk defect locations and may result is loss of user data and system downtime at a future point in time.

The RDM program will then extract information about the disk from the controller's EEPROM and display it on the console and ask:

```
Is this information correct [Y] ?
```

If these values are incorrect for the disk to be used you must run the SLEW program and change the controller's EEPROM parameters before you can recover the disk's defect information.

The RDM program will proceed to read the defect map from the disk into memory. If there are disk errors during the process the RDM program will display information about the error and retry until the defect information is recovered successfully. If the defect information is not legitimate the RDM program will inform you of the error and terminate. If the defect information is legitimate the RDM program will display each defective block in terms of cylinder, track, and sector. When the entire disk has been processed the program will display the total

number of defects found.

Once the disk has been processed the RDM program will write the defect information to either a DFS disk or MT format tape. The information will be placed in a file named:

```
<hda-serial-number>.DEFECTS
```

This file is required to format a disk with the RECOVERY program. Once the disk has been formatted the defect information recovered with RDM is no longer available. The file generated by RDM is the only source of this data and should be retained until the disk is no longer in use. The RDM program will ask about writing the defect information to either a file or tape. Answer the questions appropriately. If the RDM program was invoked from a disk it will terminate. If it was booted from tape it will restart; crash the machine to terminate it.

1.2.26. RECOVERY

The recovery program is used to prepare disks which are to be used with an R1000. The process consists of several steps which are listed here in the order required. All of these steps must be performed prior to using the disk in an R1000 system.

1. **Formatting**. The formatting process writes information onto the disk which is needed by the controller to transfer data. The format of this information can be found in Appendix A of the Spectra-Logic 121 manual. The RECOVERY program uses the controller's format drive command to format the disk. Formatting is optional when invoking the RECOVERY program unless the specified disk has no labels.

2. **Flagging bad blocks**. An integral part of building a disk for an R1000 processor is to build a bad block map which records the location of each defective area on the disk. The R1000 will not attempt to read or write to any block recorded in the bad block map. In addition to building the bad block map the RECOVERY program re-formats each bad block with the BAD SECTOR bit set in the format data. This will cause a BAD SECTOR error if any system software attempts to access these bad blocks. All bad blocks will be flagged when the RECOVERY program is invoked. This insures that blocks added to the bad block map by the system get flagged.

3. **Surface Analysis**. The surface analysis portion of the RECOVERY program is optional unless the disk was just formatted. The surface analysis is intended to insure the reliablity of the disk. From one to three passes are allowed, each taking several minutes (45 for a Fujitsu 2351 EAGLE). Each pass consists of writing to every block on the disk and then reading back each block and checking the data. A pass consists of both a write and read phase. If the disk has just been formatted the write phase of the first pass is eliminated because the process of formatting writes the data.

4. **Writing defect map**. The defect map contains information about each defective block on the disk. This data is stored on cylinder zero and is pointed to by the shared label (see below). As many as 2048 defects may be recorded with the current defect map format. The defect map is always written to the disk.

5. **Boot Label**. The boot label is located on block 1 (cylinder 0, head 0, sectors 2-3) of every disk. The boot label contains pointers to DFS files which contain the I/O Processor's Kernel and initial programs to be executed at boot time. An empty boot label is always written to the disk. It is modified as needed if a DFS is actually built on the disk.

6. **DFS Label**. The DFS label contains information about the location of the DFS directory and DFS free list. An empty DFS label is always written to the disk. It is modified as needed if a DFS is actually built on the disk. The DFS label is located on block 4 of every disk.

7. **Shared (Volume) Label**. The shared label contains information about the location of several disk structures. Some of the structures are maintained solely by the R1000 File System (RFS) and will not be mentioned here; others are shared by the DFS and the R1000 file system. These are:

   - The size of the disk. (number of cylinders, heads, sectors)
   - The location of the bad block map.
   - The location of the retarget map.
   - The location of the DFS.
   - The location of the R1000 file system.
   - The location of the read/write diagnostic portion of the disk.
   - The serial number of the disk's HDA.
   - A boolean used to indicate the presence of a DFS.
   - A boolean used to indicate the presence of an RFS.

   The shared label is always written to a disk and the portions mentioned here are only changed by the RECOVERY program. The shared label is stored using the R1000 stable storage mechanism and is located in block 2 with a copy in block 3.

8. **Building the DFS**. The DFS is only built if requested. There are several stages reported to the terminal when building a diagnostic file system. They are:

   - **Constructing free list**. The DFS retains information about free disk space as a linked list of disk extents. The free list is first constructed in memory by discarding defective blocks from the area allocated to the DFS.
   - **Writing free list**. The free list just constructed is written to the disk and its head is recorded in the DFS label.
   - **Allocating and initializing directory**. The fixed size DFS directory is allocated from the free list and initialized to be empty. At this time no files exist. Pointers to the directory are recorded in the DFS label.
   - **Allocating predefined files**. All disk structures which must have fixed disk locations or are referenced by the boot label are pre-created by the RECOVERY program. These include:
     - The disk bootstrap (located at disk block 0).
     - All bootable I/O processor kernels.
     - All bootable programs.

- All bootable file systems (series 200 only).
- The DFS error log.

9. **Loading the DFS**. After its creation the DFS may be loaded with files from an MT format tape. This step is optional.

To run the RECOVERY program you may boot it from tape or invoke it by typing:

```
CLI> x recovery
```

The program will first ask which disk drive you wish to format/build. Answer with the appropriate disk unit number. RECOVERY will then attempt to read the disk's labels and bad block map into memory. If this step fails the disk must be re-formatted and defect information read in from a tape created by the RDM program (see RDM documentation). If the labels are recovered from the disk successfully then you will be asked if the data contained in the labels should be used for the remainder of the formatting process. If you answer yes to this question then disk defect map data will be read from the disk and the disk will get built with a DFS only if it had one already.

You will be asked if you want to format the disk. Formatting the disk will destroy all data, and allow construction of a DFS on a drive which previously didn't have one.

You will be asked if you want to perform surface analysis. Surface analysis will destroy all data present on the disk. A read-only surface analysis program (CHECKDISK) can be used to check disk integrity. The RECOVERY program is not a disk test or exerciser.

Next, the disk labels will be created. If the disk had readable labels when the RECOVERY program was invoked and retained the information contained in them, that data will be used to re-build the labels. If not you will be asked to enter some information about how the disk will be used. These questions include:

```
Do you want to build a diagnostic file system on this unit [Y] ?
```

Answering yes to this question will cause space to be allocated on the disk for a DFS.

```
Enter last cylinder to be used by the DFS :
```

This question will only be asked only if the disk is to contain a DFS. The DFS will occupy all disk space between cylinder 1 and the cylinder used here. The correct answer depends on the type of disk used but should be no less than 20,000 disk blocks. It can be calculated as follows:

```
(20000 / ((H * S) / 2) + 1)

where:
H is the number of heads on the disk
S is the number of sectors on the disk
```

```
Enter first cylinder to be used for read/write diagnostics :
```

The read/write diagnostic portion of the disk starts at this cylinder and extends to the last cylinder of the disk. At least two cylinders must be allocated for read/write diagnostics. Remember that cylinder numbers start at zero, not at one; so if a disk has 842 cylinders, numbered 0 .. 841, the largest value which should be used is 840. This will cause cylinders 840 and 841 to be reserved for read/write diagnostics.

Once the disk labels have been generated the RECOVERY program is finished building the disk. If a DFS has been built on the disk you will be asked if you want to load files into the DFS. Files may be loaded from an MT format tape.

When the RECOVERY program is done, or if any unrecoverable errors occur during disk building, it will restart. The only way to terminate the RECOVERY program is by re-booting the system.

1.2.27. SAM Does Not Apply To 400

The SAM program is a programatic interface to the CDC 92185 tape drive's Structured Analysis Method of fault isolation (used with series 100 only). The SAM program may be invoked by typing:

```
CLI> x sam
```

The program is interactive and will intialize itself as follows:

```
Enter STU unit number :
```

Enter the unit number of the Streaming Tape Unit you which to test. From this point on the program will run tests which you desire and display test termination status. Some tests run forever and you will have to reset the tape drive to terminate test execution. Each SAM invocation will prompt:

```
Enter test number :
```

The program is referring to the SAM tests described in the CDC STU manual. The supported tests are 1..3, 10..26, 28..34, 37..47, 52..62, and 91, 97. Some of the tests require options. Options fall into one of four categories. They are:

```
Loop option
Bypass option
Pattern option
Speed option
```

If the test you have chosen requires options you will be prompted for them. The SAM program will then execute the selected test. Several tests do not terminate. Several other tests may not terminate if their loop option has been so set. In any case the SAM program will wait for test termination, display the termination status, and again prompt for a test to execute. To terminate the SAM program type control-C to this prompt.

1.2.28. SCAN

The SCAN program is used to search a group of text files for a given string. It is executed by typing:

```
CLI> scan
```

1.2.28.1. FIND

The FIND command requires at least one string argument. This argument is a filespec to search for a key provided via the /KEY= switch.

Switch	Description
/KEY=	Used to specify the search key.

Figure 1.16 - SCAN FIND Switches

Examples:

```
CLI> scan
SCAN> find/key=xyzzy file1,file2,*file3*
```

This command will scan FILE1, FILE2 and all files whose names contain the string FILE3. Each occurrence of XYZZY in any of these files will be displayed on the console.

1.2.29. R1000 Series 200 Models 10/20/40 PROM Debugger

The Series 200 IOC contains a 68020-based I/O Processor which replaces the PDP-11/24 used in the Series 100. To assist in various hardware and software debugging, a ROM-based, machine-level, debugger is provided. The debugger is primarily for use in manufacturing and development.

To make use of the debugger one must have fairly complete knowledge of the 68020 run-time model. The debugger's prompt is the "@" character. No type-ahead is allowed when entering debugger commands. To invoke the debugger you must place the R1000 in INTERACTIVE mode and press the BREAK key on the operations console. Then select option 3 in the menu. The debugger will be invoked immediately. Commands consist of one, two, or three characters. No carriage return is needed or allowed. Some command allow arguments which are always numbers or expresions. The radix for input and ouput may be changed. The initial radix is always 16.

Series 200 PROM Debugger Commands	
Command	Description
SD	State Display (dump all state)
RDn	Open Data register n
RAn	Open Address register n
SP	Open Stack Pointer (as defined by the PSW)
USP	Open the User Stack Pointer
ISP	Open the Interrupt Stack Pointer
MSP	Open the Monitor Stack Pointer
SR	Open the Status Register (displayed by fields)
VBR	Open the Vector Base Register
PC	Open the Program Counter
ICCR	Open the Instruction Cache Control Register
ICAR	Open the Instruction Cache Address Register
XSFC	Open the Source Function Code register
XDFC	Open the Destination Function Code register
RB	Re-Boot the IOP
RES	Software reset the IOP
expr$I	Set input radix to "expr"
$I	Display input radix
expr$O	Set output radix to "expr"
$O	Display output radix
expr$G	Set PC to "expr", and Go
$G	Go using current value of PC
expr$S	Single step through "expr" instructions
$S	Single step through 1 instruction
expr$B	Set breakpoint at address "expr"
$B	Display breakpoint list
expr$D	Delete breakpoint at address "expr"
$D	Delete all breakpoints
expr/	Open longword at address "expr"
expr\	Open word at address "expr"
expr|	Open byte at address "expr"
expr'	Open ascii character at address "expr"
^	Open previous storage unit
<LF>	Open next storage unit
<CR>	Close location
v1,V2/	Display v2 longwords starting at address v1
v1,v2\	Display v2 words starting at address v1
v1,v2|	Display v2 bytes starting at address v1

v1,v2'	Display v2 ascii characters starting at address v1
=	Display last value
expr=	Display "expr"

Figure 1.17 - Series 200 PROM Debugger Commands

Key:

```
    n           Register number 0 .. 7
    expr        An expression is a combination of numbers and
           operators.  All operators are evaluated left to
           right.  No operator precedence exists.  Allowed
           operators are:

           +    two's-complement 32-bit addition
           -    two's-complement 32-bit subtraction or
                two's-complement 32-bit negation
   32-bit one's-complement

           Where ever a number is allowed a character string
           may be used.  Character strings are enclosed in
           quotes.  In addition the "." character may be
           used in place of a number to represent the value
           of the address of the last location opened.

Examples

@123=00000123
@-123=FFFFFEDD
@~123=FFFFFEDC
@.=FFFFFEDC
@.-2=FFFFFEDA
@"ABC"=00414243
@-1+2-3+4=00000002
```

1.2.30. SLEW Does Not Apply on 400

The SLEW program is used to re-configure a Spectra-Logic SL-121 or SL-121+ tape/disk controller for use in an R1000. The controller uses an EEPROM to record several dozen options and these options are updated via SLEW. To invoke the SLEW program you may boot it from tape or type:

```
CLI> x slew
```

The program will initialize itself and then prompt:

```
Disk/Tape Controller Number :
```

Enter the controller number of the board you wish to modify. Controllers are related to devices as follows:

```
Disk     Tape        Controller
0- 3     0- 3        0
4- 7     4- 7        1
8-11     8-11        2
12-15                3
```

For the SLEW program to correctly modify a controller's EEPROM the controller must have at least one disk drive attached and on line; this is a controller limitation. The SLEW program will neither read nor write to any disks or tapes.

Once SLEW knows the controller number it begins a menu mode of operation. There are currently five menu options as follows:

```
1 => Write and verify EEPROM
2 => Verify EEPROM
3 => Display EEPROM location
4 => Modify EEPROM location
5 => Exit
```

Write and Verify EEPROM

This option allows the controller to be configured for any combination of drive types. You will be prompted to define the drive type for all 4 possible drives (0 to 3) which the controller board controls. A sample SLEW session is:

```
CLI> x slew
Disk/Tape controller number : 0

OPtions are:
    1 => Write and verify EEPROM
    2 => Verify EEPROM
    3 => Display EEPROM location
    4 => Modify EEPROM location
    5 => Exit
Enter option : 1

Enter information for unit 0
Drive type are :
    1 - Fujitsu 2351 (Eagle)
```

```
    2 - Fujitsu 2361 (Eagle XP)
    4 - Fujitsu 2333 -- 3inch, 337MB
    5 - Fujitsu 2344 -- 8inch, 690MB
     .
     .
Enter drive type : -- Select the number of the dirve from above

Enter information for unit 1
   .
   .
Is EEPROM write enabled (SW-4 open) [Y] ?
-- This switch is located on the edge of the disk/tape controller
-- board, in the middle, and can easily be accessed on Series 200
-- systems by opening the I/O cage door and reaching in.  On
-- Series 100 systems, the IOP needs to be pulled out far enough
-- to expose this portion of the board.
```

When done, make sure to set the EEPROM write enable switch back to CLOSED, and to then reboot using the *white button* reset. Failure to reset the system in this manner will result in the controller board using the older original values.

Command	Description
Write and verify	This is the option used most frequently. It updates the contents of the entire EEPROM based on questions asked of the user. The questions are asked for each of the four possible disks attached to the controller. Answer the questions carefully. Incorrect responses may prevent the system from booting after running SLEW. Once all disk drive information has been entered the SLEW program will ask: `Is EEPROM write enabled (SW-4 open) [Y] ?` The SL-121 and SL-121+ controllers have a write-protect feature. If SW-4 on the edge of the controller board is in the closed position the SLEW program cannot write to the EEPROM. Before you answer this question make sure that the switch is OPEN. If you answer 'N' to this question the SLEW program will not try to write to the EEPROM. You may use this as an escape back to the menu. Writing the EEPROM takes several seconds. It is followed by a short verify phase. Once the EEPROM has been updated, close SW-4 to write protect the data. When done with SLEW and after the different drive(s) have been installed, ALWAYS reboot the machine using the *white button* in order for the new SLEW values to take affect. Failure to do this will result in disk drive errors due to a drive of different SLEW parameters being controlled using the old SLEW parameters.
Verify EEPROM	The verify option allows the user to insure that the EEPROM checksum is correct. It does not check to insure that the data contained in the EEPROM is valid.
Display EEPROM location	The display option allows the user to examine the contents of a single EEPROM location. It is most useful for internal debugging of new EEPROM values.
Modify EEPROM location	The modify option allows the user to change the contents of a single EEPROM byte and modify the EEPROM checksum. This option is for internal use only and should not be used without a through understanding of the controller and the R1000 I/O processor kernel.
Exit	The exit option terminates the SLEW program.

Figure 1.18 - SLEW Commands

1.2.31. STARTER

The STARTER program is invoked automatically following system crashes. It is responsible for determining the course of action following a wide variety of problems. The STARTER will invoke one of the following programs depending on circumstances surrounding the crash:

```
LOADER
RDIAG
CRASHDUMP
CLI
```

The STARTER should not be invoked manually.

1.2.32. STAT

The STAT program displays directory utilization statistics for the DFS. In addition it checks for disk allocation errors and, optionally, may compact the disk. The STAT program is primarily intended for internal use. To invoke the program type:

```
CLI> x stat
```

The program will display various messages about the state of the DFS disk structures. If any error messages are displayed they should be resolved prior to using the R1000. Failure to correct DFS errors may result in user data loss and system downtime.

Interpretation of the information displayed by the STAT program requires a thorough knowledge of DFS disk structures.

1.2.33. TAPEX

The TAPEX program is a DFS-based tape exerciser. It is intended to be used for quick tape subsystem integrity testing as well as long-term reliability testing. To invoke the program type:

```
CLI> x tapex
```

The program will size the system and query the operator about testing each existing tape drive which is on-line. After these questions the exerciser will begin testing. Once testing has begun the exerciser may be stopped by typing control-G. Any other character will cause the program to display status information about each drive. This status information includes the time the test was started, the current time, total number of bytes transfered, total hard errors, total soft errors, and data errors.

The exerciser does transfers in the following manner:

1. Select a random data pattern and write a record of random length. This step is repeated a random number of times between 16 and 63.

2. Backspace a random number of records between 1 and the number of records just written.

3. Read and check the data of the records just backspaced over.

These three steps are repeated for each unit with all units running in parallel until the test is stopped.

1.2.33+ TOMBSTONE

The TOMBSTONE program is used to aid debugging R1000 microcode and hardware problems. The program will allow you to display information captured to disk when an R1000 cycles. To invoke the tombstone program:

CLI> x tombstone

```
0 => Exit
1 => Display tombstone file 1
2 => Display tombstone file 2
3 => Display tombstone file 3
4 => Display tombstone file 4

Enter option : 1

Analysis of tombstone 1 dated 09:21:48 24-JUN-99

Options are:

0 => Return to main menu
1 => Show all
2 => Show last console output
3 => Show Crash Classification
4 => Show restart output
5 => Show trace
6 => Show Cpu State
7 => Show queues
8 => Show IOP Kernel version & cluster ID
```

1.2.34. TRACE

The TRACE program is used to aid debugging R1000 microcode and hardware problems. It displays the contents of the R1000s trace rams which provide microcode execution history for the last 1024 clocks on a Series 100 or 2048 clocks on a Series 200 R1000 processor. The trace program can display data either from a CRASHDUMP which has been re-loaded into an R1000 or from a crashed machine. To invoke the trace program for CRASHDUMP data type:

```
CLI> x trace
```

To invoke the trace program for a live machine you must have previously stopped the machine using the experiment monitor. Then from EXPMON type:

```
EM> trace
```

In either case the trace program will load microcode execution history from the appropriate place into its memory buffer and display it on the console. The program is screen-oriented and provides interactive help if you type a "?".

1.2.35. UPDATE_EEPROM

This program is a DFS-based EEPROM programmer for use on the Series 200 IOC. The program provides the ability to

- Program a particular EEPROM
- Use the NOVRAM position on the IOC as an EEPROM programmer
- Do a simple write/read test of a specified EEPROM
- Verify the checksum in a specified EEPROM

To invoke the program type:

```
CLI> x update_eeprom
```

The program is menu-driven and self-prompting.

There are 4 EEPROMs on the IOC at locations K21, K19, K17, and K15. Of these, the EEPROMS in locations K21, K19, and K17 contain selftest programs, bootstrap programs, and utility programs used to boot the DFS. The EEPROM at K15 is the 'NOVRAM' that holds the board serial number, etc., placed there with NOVRAM and other data like the remote phone number placed there by INITIOA. The EEPROMs are 8192 bytes long [0..8191]. The program EEPROMs all have a common format:

```
[0000..8185]    program space
[8186]          reserved for write/read test
[8187..8189]    date code yymmdd
[8190]          coded location [17,19,21]
[8191]          checksum
```

At startup, the selftest program performs a verify operation on the checksums of the program EEPROMS. If there is an error, a message is displayed on the console and the red LED IOP ERROR is turned on. The only action here that will do anything is the WHITE BUTTON.

In principal, future program changes to the programs in these EEPROMS will be distributed on tape and this program will be used to place that new software in the EEPROMS. There will be three files of data associated with the three program EEPROMS

```
SELFTEST.HEX    ( in the EEPROM at K21 )
BOOT.HEX        ( in the EEPROM at K19 )
UTILITIES.HEX   ( in the EEPROM at K17 )
```

If the user specifies option 1 (Update EEPROM), the program will move the data from the specified file into the proper EERPOM. In the event the user wants to use the IOC as an EEPROM programmer, using option 2 will move the data from the file to the EEPROM in location K15.

2. Kernel Commands

2.1. Overview

This is the Kernel command level. Commands preceeded by a '*' are privileged commands, and can only be executed while in the privileged mode of kernel operation (see ENABLE_PRIV_CMDS).

```
 BATCH                      CHANGE_GC_THRESHOLDS      CLEAR_PROFILE
 CLEAR_PROFILES             DISABLE_JOB               ENABLE_JOB
 ENABLE_PRIV_CMDS           JOB                       JOBS
 JOB_NAME                   JOB_NAMES                 JOB_MTS
 JOBS_MTS                   LOAD                      MTSQ
 NOOP                       PROFILE                   PROFILES
 QUIT                       SET_MTS_PARAM
SET_TASK_FILTER
 SHOW_BAD_BLOCKS            SHOW_CONFIGURATION_BITS
SHOW_DISK_SUMMARY
 SHOW_ERROR_LOG             SHOW_GC_STATE
SHOW_MEMORY_UTIL
 SHOW_MTS_PARAMS            SHOW_NEXT_SNAPSHOT        SHOW_PORT_INFO
 SHOW_TASK_FILTER           SHOW_TASK_STATES
SHOW_VOLUME_SUMMARY

 TIME                      *ABORT_TASK
*BUILD_NEW_SYSTEM
*CHANGE_GHOST_LOGGING      *CREATE_CG_VPS
*CREATE_EMPTY_SPACE
*CREATE_VP                 *DEFAULTS                 *DELETE_SPACE
*DISABLE_PRIV_CMDS         *DISABLE_SUB_LOGGING
*ENABLE_SUB_LOGGING
*ENTER_DEBUG_CONTEXT       *FIND_BLOCK_REFS
*FIND_DISLOCATED_BLKS
*GO_BACK_IN_TIME           *HOGS                     *LMR
*LMW                       *PARTIAL_STARTUP
*REMEMBER_DEFECT
*ROUST                     *SET_SUB_BUFFER_SIZE      *SET_SUB_FIELDS
*SHOW_ALL_SPACES           *SHOW_CACHED_SPACES
*SHOW_CATALOG_STRUCT
*SHOW_CONFIGURATION        *SHOW_DEFAULTS
*SHOW_GC_FOOTPRINT
*SHOW_GHOST_LOG            *SHOW_HASH                *SHOW_MEMORY
*SHOW_SPACE_INFO           *SHOW_SPACE_STRUCT
*SHOW_SUB_FIELDS
*SHOW_SUB_TRACE            *SHOW_TAGS                *SHOW_UCODE_REG
*SHOW_VOLUME_STRUCT        *SHOW_VPS                 *SHUTDOWN
*START_ENVIRONMENT         *START_NETWORK_IO
```

```
*START_VIRTUAL_MEMORY
*TAKE_SNAPSHOT              *TRACE
*TRAVERSE_VM_STRUCT
*ZAP_BROKEN_SPACES          *ZERO_BLOCK
```

2.2. BATCH

This command will display what jobs are currently running in the batch system streams (queues).

Example

```
 *Kernel: batch
Stream 1       2:00
Stream 2       58:00
   225  51:03
Stream 3       50:00
   231  47:00
   234  46:56
   222  46:38
   233  45:42
```

2.3. CHANGE_GC_THRESHOLDS

Example

```
Kernel: change_gc_thresholds
VOLUME_NUMBER [1]:
THRESHOLD [START_COLLECTION]:
REMAINING CAPACITY (%) [10]:

Kernel: change_gc_thresholds
VOLUME_NUMBER [1]: 4
THRESHOLD [SUSPEND_SYSTEM]: xxx
EXPECTED VALUES ARE:
 START_COLLECTION  RAISE_PRIORITY    STOP_JOBS          SUSPEND_SYSTEM
 SPACE_04
THRESHOLD [SUSPEND_SYSTEM]:
REMAINING CAPACITY (%) [8]:
```

2.4. CLEAR_PROFILE

Example

```
Kernel: clear_profile
VPID [4]:
```

2.5. CLEAR_PROFILES

Example

```
Kernel: clear_profiles
```

2.6. DISABLE_JOB

Example

```
Kernel: disable_job
VPID [4]: 222
```

2.7. ENABLE_JOB

Example

```
Kernel: enable_job
VPID [222]: 223
```

2.8. ENABLE_PRIV_CMDS

Example

```
Kernel: enable_priv_cmds
You are enabling a set of commands which must be used
with extreme care.  They should be used only by
knowledgeable support personnel.  These commands can
easily crash/hang the machine; some can competely tråsh
the state of the machine such that you must recover the
machine from backup tapes.
Proceed [FALSE]: true
Password: secret
*Kernel:
```

2.9. JOB

Example

```
 Kernel: job
VPID [4]: 222
 Job Pri Stat CPU%  ModCt Cache   Disk  PgLim DskWts D/S JSegSz WsSiz WsLim
---- --- ---- ---- ------ ----- ------ ------ ------ --- ------ ----- -----
 222   0 I,DT    0      4     8     11   8000    130   0      5     5     0
```

2.10. JOBS

Example

```
 Kernel: jobs
Threshold [2]: xx
EXPECTED VALUES ARE:  0 ..  2147483647
Threshold [2]: 1
 Job Pri Stat CPU%  ModCt Cache   Disk  PgLim DskWts D/S JSegSz WsSiz WsLim
---- --- ---- ---- ------ ----- ------ ------ ------ --- ------ ----- -----
   4   0 R,AT    1   4104  9195  14150  65536 604891   0   1446 10997 11000
   5   0 R,AT    0     12    80     82  65536   2127   0     93    36   200
 183   0 I,AT    0     24    43     57   8000    184   0     68   146    50
 220   6 I,AT    0      2     1      9  16000    427   0      3     1     0
 222   0 I,DT    0      4     8     11   8000    130   0      5     6     0
 223   6 I,AT    0      2     1      9  16000   2240   0     16     0     0
 224   6 R,OE    0      1     3      0  16000    650   0     45    72    75
 227   6 I,AT    0      2     2      8  16000    843   0     32     0     0
 228   0 I,SV    0      3     8      9   8000    970   0     12    25    75
 229   0 I,SV    0      9    61     58   8000   7290   0   1592    72    75
 231   0 I,DT    0     47     8    172   8000    223   0      3     0    50
 232   6 I,CE    0      1     1      5  16000     25   0      4     0     0
 233   0 I,DT    0     22    44     61   8000    288   0     10     1    50
 247   6 I,CE    0      1     6      6  16000    152   0     15    36   150
 248   6 I,CE    0     42    80    300  16000   1192   0    102    74   150
 249   6 I,CE    0      1     5      6  16000     37   0      5    18   150
 250   6 I,CE    0      1     1      6  16000      4   0      0     0     0
 251   6 I,CE    0      1     0      6  16000      4   0      0     0     0
 252   6 I,CE    0      1     0      7  16000    429   0      7    10   150
 253   6 I,AT    0      2    10      1   8000   1056   0      0     1     0
 254   0 I,SV    0     21     2     66   8000     72   0     73     0     0
 255   6 I,CE    0      1     6      6  16000    132   0     65     7    10
```

2.11. JOB_NAME

Example

```
 Kernel: job_name
VPID [222]:
 Job CPU%      Root   Job Seg   Name
---- ---- -------- ---------  --------------------------
 222    0    4A8DE  10FB1503 \Mail_Check
```

2.12. JOB_NAMES

Example

```
 Kernel: job_names
Threshold [1]:
 Job CPU%      Root   Job Seg   Name
---- ---- -------- ---------  --------------------------
   4    4        0  14625502 System
   5    0        0  14626902 Daemons
```

```
 183    0    360B7  1499A902 SMP.DELTA.GURU % OP.INTERNAL_SYSTEM_DIAGNOSIS
 186    0        0  14841902 <?>
 188    0        0         0 <?>
 190    2        0  162B0101 [SMP.S_1 Editor]
 202    0        0  15F38901 [GZC.S_1 Command]
 206    0        0  1477E102 [LAP.S_1 Command]
 208    0        0  10FEE103 [GZC.S_1 Editor]
 209    0    32CD1  113B2100 Archive Server
 210    0   10E8D2  16057501 Queue Server
 211    0        0  147F8D02 [GZC.DESIGN Command]
 212    0    1A4D4  112DF900 Mail Transceiver
 213    0        0  11565900 [MLV.S_1 Command]
 222    0    4A8DE  10FB1503 \Mail_Check
 224    0        0  14928502 [SMP.S_1 Command]
 228    0    2B4E4  14652102 (Ftp Server)
 229    1    3E4E5  15E69101 Mail Dispatcher
 231    0    1C8E7  10F1F103 Design Facility (Rev3_2 release)
 232    0        0  112D5D00 *Login: 247
 233    0    428E9  15E63101 Mail Oe
 234    0    558EA  14657102 Registration job for PDL named PDL_2167
 235    0    20CEB  10F1ED03 Console Command Interpreter
 241    0        0  15E4D101 *Login: 246
 242    0        0  10F1A503 Print_Spooler
 252    0        0  112CED00 *Login: 16
 254    0    2A8FE  1462B902 "!COMMANDS.INTERNAL.DEC20.REV9_1_SPEC.UNITS.SER
```

2.13. JOB_MTS

Example

```
 Kernel: job_mts
VPID [222]:
 Job   K/S/P  Stat      CPU      CPU    Disk    Disk    WSet   WSet  Map   Run
               Age   Seconds    MS/S   DW/S   Waits    Size  Limit   To  Ratio
-------------------------------------------------------------------------------
 222  *D/I/0    54  00002.624     0.0    0.0     130      15      0  190   1.00
```

2.14. JOBS_MTS

Example

```
 Kernel: jobs_mts
 Job   K/S/P  Stat      CPU      CPU    Disk    Disk    WSet   WSet  Map   Run
               Age   Seconds    MS/S   DW/S   Waits    Size  Limit   To  Ratio
-------------------------------------------------------------------------------
   4   A/R/0  ++++  10594.728    70.8    0.0  604941   10999  11000        1.00
   5   A/R/0  ++++  05802.336     0.0    0.0    2127      36    200        1.00
 183  *A/R/0     2  00007.819    10.6    0.2     187      75    100  190   1.00
 184   T/I/6   126  00000.108     0.0    0.0       5      43      0        1.00
 190   C/R/6     2  00223.696    48.4    0.0    3250     174    150        1.00
 191   T/I/6  ++++  00013.593     0.0    0.0     573       1      0        0.99
 209  *S/I/0  ++++  00000.104     0.0    0.0      11       0      0        1.00
 210  *S/I/0  ++++  00000.094     0.0    0.0       7       0      0        1.00
 211   O/I/6  1192  00003.964     0.0    0.0      89       2     50  218   1.00
```

```
 212  *S/I/0   112  00442.230    0.0    0.0    4167     75     75          1.00
 213   O/I/6   100  00001.858    0.0    0.0      46    117     75   217    1.00
 215   C/I/6  9001  00030.487    0.0    0.0     515     94    150          1.00
 216   C/I/6     0  00022.941   61.0    0.0    1065    151    150          1.00
 217   C/I/6    83  00021.728    0.0    0.0     247    150    150          0.99
 218   C/I/6  9000  00055.883    0.0    0.0     817     19     50          1.00
 248   C/I/6  7335  00051.416    0.0    0.0    1192     74    150          1.00
 249   C/I/6  ++++  00001.438    0.0    0.0      37     18    150          1.00
 250   C/I/6  ++++  00000.046    0.0    0.0       4      0      0          1.00
 251   C/I/6  ++++  00000.026    0.0    0.0       4      0      0          1.00
 252   C/I/6  ++++  00005.782    0.0    0.0     429     10    150          1.00
 253   T/I/6  9982  00053.682    0.0    0.0    1056      1      0          0.99
 254  *S/I/0  ++++  00001.430    0.0    0.0      72      0      0          1.00
 255   C/I/6  8968  00005.236    0.0    0.0     132      7     10          1.00
```

2.15. LOAD

Example

```
 Kernel: load
Run Queue Load      => 1.16, 0.69, 0.56, 0.47
Disk Wait Load      => 0.00, 0.02, 0.04, 0.09
Withheld Task Load => 0.00, 0.00, 0.00, 0.00
Available Memory    => 18698 pages
```

2.16. MTSQ

Example

```
 Kernel: mtsq
Foreground Q
Background Q
Internal Transition Q
```

2.17. NOOP

Example

```
 Kernel: noop
```

2.18. PROFILE

Example

```
 Kernel: profile
VPID [222]: 190
 Job    Made      Made      Run       Made     Wait D   Wait C   Wait M
        Idle      Run       Total     Wait     Total    Total    Total
```

2.19. PROFILES

Example

```
Kernel: profiles
Job    Made      Made      Run       Made     Wait D    Wait C    Wait M
       Idle      Run       Total     Wait     Total     Total     Total
-------------------------------------------------------------------------
183       45        45       305        0         0         0         0
190      792       792      2374        0         0         0         0
195        6         6         7        0         0         0         0
196       11        11        20        0         0         0         0
197        2         2         4        0         0         0         0
202        5         5         5        0         0         0         0
206        6         6         7        0         0         0         0
208        2         2         4        0         0         0         0
209        0         0         0        0         0         0         0
210        0         0         0        0         0         0         0
211        5         5         5        0         0         0         0
212      171       171       408        0         0         0         0
213       35        35        89        0         0         0         0
215       24        24        64        0         0         0         0
216      642       643      2042        0         0         0         0
217      374       387      1547       13         0        14         0
218        2         2         4        0         0         0         0
```

2.20. QUIT

Example

```
Kernel: quit
EEDB:
```

2.21. SET_MTS_PARAM

Example

```
Kernel: set_mts_param
parameter name : help
parameter value [0]: 9
no such parameter
```

2.22. SET_TASK_FILTER

This command allows setting of filter attributes for use by the Show_Task_States command. By convention, the Set_Task_Filter command is used only from EEDB; this allows "quit", followed by EEDB "kernel" command to revert the task filter to

its default state.

See Show_Task_States, Show_Task_Filter.

Example

```
Kernel: set_task_filter
FILTER_KIND [BY_BLOCK_CONDITION]: xx
EXPECTED VALUES ARE:
 BY_BLOCK_CONDITION  BY_WAIT_STATE       BY_VPID
FILTER_KIND [BY_BLOCK_CONDITION]:
SELECTION_KIND [JUST_ONE]: xx
EXPECTED VALUES ARE:
 JUST_ONE   EVERY_ONE  PROMPT
SELECTION_KIND [JUST_ONE]:
BLOCK_CONDITION [SPARE_21]: xx
EXPECTED VALUES ARE:
 UNBLOCKED                    DECLARING_MODULE
 AWAITING_ACTIVATION          ACTIVATING_MODULE
 ACTIVATING_TASKS             AWAITING_TASK_ACTIVATION
 AWAITING_CHILDREN            TERMINABLE_AT_END
 BLOCKING_ON_ENTRY            DELAYING_ON_ENTRY
 ATTEMPTING_ENTRY             DELAYING
 ABORTING_MODULE              TERMINATED
 IN_FS_RENDEZVOUS             IN_WAIT_SERVICE
 DELAYING_IN_WAIT_SERVICE     BLOCKING_IN_ABORT
 DELETED                      ABORTED_WHILE_IN_MTS
 IN_MTS_RENDEZVOUS            SPARE_21
 SPARE_22                     SPARE_23
 BLOCKING_ON_ACCEPT           BLOCKING_ON_SELECT
 DELAYING_ON_SELECT           AWAITING_CHILDREN_IN_SELECT
 TERMINABLE_IN_SELECT         SPARE_29
 SPARE_30                     SPARE_31
BLOCK_CONDITION [SPARE_21]:
show tasks with this block condition [FALSE]: xx
EXPECTED VALUES ARE:
 YES    TRUE   NO     FALSE
show tasks with this block condition [FALSE]:
```

2.23. SHOW_BAD_BLOCKS

This command is used to display bad block information for a specified disk drive. Most disk drives have a certain number of defects when shipped from the factory. These are identified by the factory and provided with the drive so that those blocks are not used. During the drive's lifetime, more bad blocks will start to appear.

The Environment, when it encounters a new disk block which generates errors, will automatically add that block to the manufacturers list maintained on the disk drive, and *Retarget* that block to a known good block on disk. Thus, whenever a read or write is attempted to the origional block, the retarget block information is used and redirects the read/write to the new block.

The Manufacturers_And_System option will display all bad block information for the drive, in sorted order. The Retarget option will display just the set of

"retargeted" bad blocks, in sort order. Since a retargeting event generally causes the block to be entered in the bad block list immediately, retargeted blocks will show up in both lists.

Example

```
 Kernel: show_bad_blocks
VOLUME_NUMBER [1]:
KIND [MANUFACTURERS_AND_SYSTEM]: xx
EXPECTED VALUES ARE:
 MANUFACTURERS_AND_SYSTEM  RETARGET
KIND [MANUFACTURERS_AND_SYSTEM]: retarget
Kernel: show_bad_blocks
VOLUME_NUMBER [1]:
KIND [RETARGET]: manufacturers_and_system
blocks => 322 .. 322 || sectors => 0, 13, 20 .. 0, 13, 20
blocks => 425 .. 425 || sectors => 0, 17, 34 .. 0, 17, 34
blocks => 715 .. 715 || sectors => 1, 9, 38 .. 1, 9, 38
blocks => 1380 .. 1380 || sectors => 2, 17, 24 .. 2, 17, 24
blocks => 1492 .. 1492 || sectors => 3, 2, 8 .. 3, 2, 8
blocks => 1860 .. 1860 || sectors => 3, 17, 24 .. 3, 17, 24
blocks => 2340 .. 2340 || sectors => 4, 17, 24 .. 4, 17, 24
blocks => 2820 .. 2820 || sectors => 5, 17, 24 .. 5, 17, 24
blocks => 3300 .. 3300 || sectors => 6, 17, 24 .. 6, 17, 24
blocks => 3780 .. 3780 || sectors => 7, 17, 24 .. 7, 17, 24
```

2.24. SHOW_CONFIGURATION_BITS

Display the Boot/Crash/Maintenance Options, and some other information about power state of the processors.

Example

```
Kernel: show_configuration_bits
 IOP 0 POWER ON
 CPU 0 POWER ON
 OPERATOR MODE => INTERACTIVE
 KERNEL DEBUGGER AUTO BOOT => TRUE
 KERNEL AUTO BOOT          => TRUE
 EEDB AUTO BOOT            => TRUE
 KERNEL DEBUGGER WAIT ON CRASH    => TRUE
 KERNEL DEBUGGER DIALOUT ON CRASH => TRUE
 DIAGNOSTIC MODEM CAN DIALOUT => FALSE
 DIAGNOSTIC MODEM CAN ANSWER  => TRUE
```

2.25. SHOW_DISK_SUMMARY

There are 3 parts to the disk summary display.

1. A table
2. A list of in progress IO's

3. and some debugging information.

The table contains the following information, described by column.

- **Vol**. Stands for volume number
- **unt**. Stands for unit number. by convention unit i = volume i+1. But this correspondence is actually driven by the unit numbers selected at the drive, and can therefore be different.
- **Q Len**. Gives the number of blocks currently queued, but not yet issued to the iop.
- **IOP Len**. Gives the number of io requests which have been issued and not yet serviced by the iop.
- **Total Reads**. Gives the number of blocks read from the unit, since boot.
- **Total Writes**. Gives the number of blocks written to the unit, since boot.

The remaining columns displays error counts, since boot.

Errors

Seek Error, should be obvious. A "soft ecc" error is a data ecc error that was correctable. These cause blocks to be retargeted. A "hard ecc" error is a data ecc error that was not correctable. An "unrecoverable" error is any error which prevents the completion of the requested io; this includes hard data ecc errors; these generally hang the machine and require use of the manual disk error recovery procedure.

The list of in progress IOS might simply say "no disk io in progress". Otherwise, if display one line for each block which is currently involved in disk io (includes both queued blocks and requests waiting for response from the iop). Each line gives the "block address"; (3, 1057) means volume 2, block 1057. Note that a translation is required to get from the block number to the physical <cyl,trk,sector> address. Each line gives the "page address"; (1023, data, 259, 10234) means vpid 1023, segment kind "data", segment number 259, page number 10234. Each lines gives an "arrow" indicating the direction of the IO.

Example

```
 Kernel: show_disk_summary
DISK STATUS SUMMARY

          Q   IOP    Total      Total    Seek  Soft  Hard    Un   Total
 Vol  Unt Len  Len    Reads      Writes   Errs   Ecc   Ecc Recov   Errs
-------------------------------------------------------------------------
   1    0   0    0   190106       77972     0     0     0     0      0
   2    1   0    0   393305      127880     0     0     0     0      0
   3    2   0    0   243519       83758     0     1     0     0      1
```

```
   4    3    0    0     182910     124274     0     0     0     0     0

no disk IO in progress

Debugging information:
Ready_Volume mask => 0
Busy_Event_Page => ( 1023, DATA, 259, 241)
Volume_Offline_Event_Page => ( 1023, DATA, 259, 242)
```

2.26. SHOW_ERROR_LOG

Let "current log" denote that portion of the error which is stored by the kernel and not yet copied into a file in "!machine.error_logs". This command is used to display portions of the current log. Asks for line numbers. these are relative to the first line of the current log. The range of lines is displayed in chronological order when the first line number is less than the last line number. When last is greater than first, displays the log is reverse order.

The format of the entries is defined elsewhere in the documentation.

Example

```
 Kernel: show_error_log
first entry => 1; last entry => 180
FIRST [180]:
LAST [170]:
09:49:06 --- Ethernet Controller_Status EXOS CODE 0003 rxmt #1, 2 sec
09:40:10 !!! Job_Manager Bad_Job_Id Id =  253, Count = 2 Names = 2348FD 2344FD
09:40:09 !!! Job_Manager Bad_Job_Id Id =  199, Count = 2 Names = 97CC7
09:40:08 !!! Job_Manager Bad_Job_Id Id =  227, Count = 2 Names = 24E0E3
09:40:07 !!! Job_Manager Bad_Job_Id Id =  188, Count = 2 Names =
09:40:06 !!! Job_Manager Bad_Job_Id Id =  186, Count = 2 Names =
09:40:05 !!! Job_Manager Bad_Job_Id Id =  192, Count = 2 Names =
09:40:04 !!! Job_Manager Bad_Job_Id Id =  205, Count = 2 Names =
09:40:03 !!! Job_Manager Bad_Job_Id Id =  198, Count = 2 Names =
09:40:02 !!! Job_Manager Bad_Job_Id Id =  219, Count = 2 Names =
09:40:01 !!! Job_Manager Bad_Job_Id Id =  214, Count = 1 Names =
09:40:03 !!! Job_Manager Help Me Mr. Wizard!
```

2.27. SHOW_GC_STATE

Example

```
Kernel: show_gc_state
DISK daemon is not running
```

2.28. SHOW_MEMORY_UTIL

Example

```
Kernel: show_memory_util
MEMORY_SIZE => 32768

   ATTRIBUTE     CTL     TYP      Q     DATA     IMP     CODE    TOTAL
-------------------------------------------------------------------------
       DIRTY    2294    1051     48     7557     931      837    12719
   WRITEABLE    3789    1758     48    14239    1615      145    21595
       WIRED     723     396     60     1549     293      830     3852
  IN_TRANSIT       0       0      0        0       0        0        0
   PERMANENT       0       0      0     7448       0     3593    11041
 SNAPSHOTABLE      0       0      0      711       0        0      711
  RECLAIMABLE      0       0      0        0       0        0        0
       TOTAL    4013    1763     75    18844    1615     4569    30880

  ATTRIBUTE   MIN  MAX
----------------------
       DIRTY    0   13
       WIRED    0    7
 RECLAIMABLE    0    0
```

2.29. SHOW_MTS_PARAMS

Example

```
 Kernel: show_mts_params
Cpu_Scheduling    : Enabled
Disk_Scheduling   : Enabled
Memory_Scheduling : Enabled

Percent_For_Background         : 20%
Min_ and Max_Foreground_Budget :-250 .. 250 milliseconds
Withhold_Run_Load              : 130
Withhold_Multiple_Jobs         : FALSE

Environment_Wsl           : 11000 pages
Daemon_Wsl                : 200 pages
Min_ and Max_Ce_Wsl       : 150 .. 500 pages
Min_ and Max_Oe_Wsl       : 75 .. 750 pages
Min_ and Max_Attached_Wsl : 50 .. 4000 pages
Min_ and Max_Detached_Wsl : 50 .. 4000 pages
Min_ and Max_Server_Wsl   : 75 .. 1000 pages
Min_Available_Memory      : 1024 pages
Wsl_Decay_Factor          : 50 pages/5 seconds
Wsl_Growth_Factor         : 50 pages/100 milliseconds
Page_Withdrawal_Rate      : 1*640 pages/second

Min_ and Max_Disk_Load : 200 .. 250

Foreground_Time_Limit   : 1800 seconds
Background_Streams      : 3
Strict_Stream_Policy    : FALSE
Stream_Time and _Jobs 1 : 2 minutes, 3 jobs
Stream_Time and _Jobs 2 : 58 minutes, 1 job
Stream_Time and _Jobs 3 : 50 minutes, 0 jobs
```

2.30. SHOW_NEXT_SNAPSHOT

Example

```
 Kernel: show_next_snapshot
SNAPSHOT_NUMBER => 3815
```

2.31. SHOW_PORT_INFO

Example

```
 Kernel: show_port_info
PORT_MANAGER:     INPUT   OUTPUT
                  -----   ------
    BYTES....     51808  1596718
    PACKETS..     96473    88001
PORT_ID [0]: 16
OUTPUT: CLIENT => 677054; IOP IS BUSY
INPUT: CLIENT => 668862
Kernel: show_port_info
PORT_MANAGER:     INPUT   OUTPUT
                  -----   ------
    BYTES....     51857  1597240
    PACKETS..     96539    88051
PORT_ID [16]: 33
OUTPUT: NO CLIENT REGISTERED
INPUT: NO CLIENT REGISTERED
```

2.32. SHOW_TASK_FILTER

This command will show the filter settings used by the Show_Task_States command. The Set_Task_Filter command is used to set these filters.

See Set_Task_Filter, Show_Task_States.

Example

```
 Kernel: show_task_filter
FILTER_KIND [BY_BLOCK_CONDITION]: xx
EXPECTED VALUES ARE:
 BY_BLOCK_CONDITION  BY_WAIT_STATE       BY_VPID
FILTER_KIND [BY_BLOCK_CONDITION]:
  want UNBLOCKED => TRUE
  want DECLARING_MODULE => FALSE
  want AWAITING_ACTIVATION => FALSE
  want ACTIVATING_MODULE => FALSE
  want ACTIVATING_TASKS => FALSE
  want AWAITING_TASK_ACTIVATION => FALSE
  want AWAITING_CHILDREN => FALSE
  want TERMINABLE_AT_END => FALSE
  want BLOCKING_ON_ENTRY => FALSE
  want DELAYING_ON_ENTRY => FALSE
```

```
  want ATTEMPTING_ENTRY => TRUE
  want DELAYING => FALSE
  want ABORTING_MODULE => TRUE
  want TERMINATED => FALSE
  want IN_FS_RENDEZVOUS => TRUE
  want IN_WAIT_SERVICE => TRUE
  want DELAYING_IN_WAIT_SERVICE => TRUE
  want BLOCKING_IN_ABORT => TRUE
  want DELETED => TRUE
  want ABORTED_WHILE_IN_MTS => TRUE
  want IN_MTS_RENDEZVOUS => TRUE
  want SPARE_21 => FALSE
  want SPARE_22 => TRUE
  want SPARE_23 => TRUE
  want BLOCKING_ON_ACCEPT => FALSE
  want BLOCKING_ON_SELECT => FALSE
  want DELAYING_ON_SELECT => FALSE
  want AWAITING_CHILDREN_IN_SELECT => FALSE
  want TERMINABLE_IN_SELECT => FALSE
  want SPARE_29 => TRUE
  want SPARE_30 => TRUE
  want SPARE_31 => TRUE
Kernel: show_task_filter
FILTER_KIND [BY_BLOCK_CONDITION]: xx
EXPECTED VALUES ARE:
 BY_BLOCK_CONDITION  BY_WAIT_STATE       BY_VPID
FILTER_KIND [BY_BLOCK_CONDITION]: by_vpid
 0 .. 1023 => TRUE
Kernel: show_task_filter
FILTER_KIND [BY_VPID]: by_wait_state
  want PACKET_ID_WAIT => TRUE
  want PORT_WAIT => TRUE
  want TAPE_WAIT => TRUE
  want SYSTEM_BOOT_WAIT => TRUE
  want VOLUME_LOW_ON_SPACE_WAIT => FALSE
  want SNAPSHOT_WAIT => FALSE
  want PORT_INPUT_WAIT => FALSE
  want PORT_OUTPUT_WAIT => TRUE
  want TAPE_INPUT_WAIT => TRUE
  want TAPE_OUTPUT_WAIT => TRUE
  want PAGE_POOL_WAIT => TRUE
  want X25_WAIT => TRUE
  want X25_CALL_WAIT => TRUE
  want X25_INPUT_WAIT => TRUE
  want X25_OUTPUT_WAIT => TRUE
  want DEVICE_ERROR_LOG_WAIT => FALSE
  want MEMORY_ECC_WAIT => FALSE
  want PACKET_ID_LIMIT_WAIT => FALSE
  want PAGE_WIRE_WAIT => TRUE
  want KERNEL_DEBUGGING_WAIT => TRUE
  want SHORT_TERM_LOCK_WAIT => TRUE
  want TCP_IP_INPUT_WAIT => TRUE
  want TCP_IP_OUTPUT_WAIT => TRUE
  want U023 .. U031 => TRUE
  want CORE_EDITOR_WAIT => TRUE
  want COMPILATION_REQUEST_WAIT => TRUE
  want ACTION_MANAGER_WAIT => TRUE
  want ENVIRONMENT_DEBUGGING_WAIT => TRUE
  want NATIVE_DEBUGGING_WAIT => TRUE
  want WINDOW_INPUT_WAIT => FALSE
  want PIPE_INPUT_WAIT => FALSE
```

```
want PIPE_OUTPUT_WAIT => FALSE
want U040 .. NO_STATE => TRUE
```

2.33. SHOW_TASK_STATES

The Show_Task_States command is used to display some attributes of "interesting" modules. There are 2 primary module attributes which are examined to determine if a module is interesting:

- The module's Virtual Process ID (VPID)
- The module's block condition

The VPID filter indicates which VPIDs are considered interesting. and the block condition indicates which block conditions are considered interesting. If a module has an interesting VPID or block condition, it will be displayed. There is a sub-attribute, **Wait State**, which is examined when the module's block condition is one of the 2 wait state block conditions. There is also a filter for this sub-attribute. The Show_Task_Filter command can be used to diplay the current setting of the filters. And the Set_Task_Filter can be used to modify the filters.

There are some additional module attributes which always make a module look interesting, regardless of filter setting. For example, *aborted* modules are always considered interesting.

Example

```
 Kernel: show_task_states
CACHE/DISK [CACHE]:
16#820F8#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 4
16#E222004#; HELD_BY_MTS; PRI 3
16#2B4E4#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 14
16#23CD4#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 14
16#32BCD8#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 4
16#A60D9#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 4
16#374B7#; UNBLOCKED; PRI 8
16#4B0DE#; UNBLOCKED; PRI 14
16#E153C04#; DELAYING_IN_WAIT_SERVICE U031; PRI 3
16#B5DDC04#; UNBLOCKED; PRI 13
16#798CB#; ABORTED; PRI 14
16#DDF6804#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 3
16#427EC04#; IN_WAIT_SERVICE X25_WAIT; PRI 1
16#F5CD7#; IN_WAIT_SERVICE TCP_IP_INPUT_WAIT; PRI 4
Kernel: show_task_states
CACHE/DISK [CACHE]: disk
Must first use the ENABLE_PRIV_CMDS command
```

2.34. SHOW_VOLUME_SUMMARY

Thresholds, garbage collection, etc is documented in the sys mgrs guide. The capacity table has a column labeled "rate blks/min". these values give the number of blocks consumed (per minute) since the last time this command was invoked. the

space_04 threshold is unused - so ignore it. the "next trigger" values give the value of unused capacity at which the next threshold will be triggered. otherwise, the display should be self explanatory.

Example

```
 Kernel: show_volume_summary
Volume Status Summary

 Vol     Total       Unused       Rate
 Num    Capacity    Capacity    Blks/Min
---------------------------------------
   1      369120      176712           0
   2      391680      101288           1
   3      391680      141456           0
   4      401280      142433           0

low space thresholds for volume 1:
    START_COLLECTION threshold at 20% (waiters exist)
    RAISE_PRIORITY threshold at 15% (waiters exist)
    STOP_JOBS threshold at 12% (waiters exist)
    SUSPEND_SYSTEM threshold at 7% (waiters exist)
    SPACE_04 threshold at 0% (no waiters)
    next trigger at 73824 blocks
low space thresholds for volume 2:
    START_COLLECTION threshold at 20% (waiters exist)
    RAISE_PRIORITY threshold at 15% (waiters exist)
    STOP_JOBS threshold at 10% (waiters exist)
    SUSPEND_SYSTEM threshold at 8% (waiters exist)
    SPACE_04 threshold at 0% (no waiters)
    next trigger at 78336 blocks
low space thresholds for volume 3:
    START_COLLECTION threshold at 20% (waiters exist)
    RAISE_PRIORITY threshold at 15% (waiters exist)
    STOP_JOBS threshold at 10% (waiters exist)
    SUSPEND_SYSTEM threshold at 8% (waiters exist)
    SPACE_04 threshold at 0% (no waiters)
    next trigger at 78336 blocks
low space thresholds for volume 4:
    START_COLLECTION threshold at 20% (waiters exist)
    RAISE_PRIORITY threshold at 15% (waiters exist)
    STOP_JOBS threshold at 10% (waiters exist)
    SUSPEND_SYSTEM threshold at 8% (waiters exist)
    SPACE_04 threshold at 0% (no waiters)
    next trigger at 80256 blocks

Debugging information:
    OUT_OF_SPACE_EVENT_PAGE_ADDR => ( 1023, DATA, 259, 504)
```

2.35. TIME

Example

3. EEDB Commands

3.1. Overview

EEDB can be run from an environment window by calling Op.Internal_System_Diagnosis, or can be used via the system console (use ^Z to get to the EEDB: prompt).

EEDB builds and maintains configurations of subsystems to be elaborated to run system programs. A number of general principals may be of interest.

A configuration is an ordered list of subsystems in the order that they are to be elaborated. Configurations are built from other configurations and share all subsystems below the branch point. For example, there is usually a MUX configuration, with OM and Dir configurations built off of it. Changing the subsystems in MUX will cause the corresponding subsystems in OM and Dir to change. There is also a copy facility that creates an equivalent configuration that is a full copy.

Most commands accept a wildcard notation. Either '+' or '*' match 0 or more characters. Wildcards are only permitted at the beginning and/or end of the argument, i.e. +.5.+ is legal, but a_+_0 won't ever match anything (or give an error message).

Subsystem names can be abbreviated using standard short names for the subsystems. For use with wildcards, the abbreviation must be followed by '.'; i.e. UAT.+ matches all Abstract_Types subsystems, but UAT+ doesn't.

Commands and some operands accept prefixes. For commands or arguments that are being specified, unique prefixes are required. For operations that display values (e.g. Abbreviations and Help), all values matching the prefix are shown as if a + were appended to the parameter.

Commands that accept lists of arguments terminate with an empty entry.

Commands and arguments can be typed on a single line, one line per token, or any combination in between. New lines will cause a prompt to be printed for the next value. Unrecognized values will produce a message; if the value is an enumeration, possible values will be printed. In either case, the prompt will be re-issued. Commands can be terminated by either ^G or ^C (must be "quoted" if running from environment).

There is a page mode that keeps output from scrolling off the terminal and allows a number of operations. See Terminal_Settings for how to set page mode, etc. Examples of the interaction with page mode:

```
-- MORE --  (n, o, q, r, s, ?) ?
N, ^G, ^C  =>  stop command
O, ^O      =>  suppress command output
S          =>  skip output between MOREs
```

3.2. Commands

A summary of EEDB commands are:

```
ABBREVIATIONS          - Print subsystem abbreviation/full-name pairs
ADD_SUBSYSTEM          - Add a subsystem to a configuration
BUILD_CONFIGURATION    - Build a configuration
COMMON                 - Highest common subsystem for two configurations
COPY_CONFIGURATION     - Copy a configuration
CHECK_CONSISTENCY      - Check consistency of database
DEFAULT_CONFIGURATION - Set the default configuration
DELETE                 - Delete a subsystem or configuration
DISPLAY                - Short form display subsystem/configuration
ELABORATE              - Elaborate a configuration
FIND_SEGMENT           - Find a segment number
HELP                   - Print a help message
INSERT_SUBSYSTEM       - No help available for INSERT_SUBSYSTEM
KERNEL                 - No help available for KERNEL
QUIT                   - Leave Command Interpreter
READ_TAPE              - Read from tape
RECLAIM_SPACE          - No help available for RECLAIM_SPACE
REMOVE_SUBSYSTEM       - Remove a Subsystem from a Configuration
REPLACE_SUBSYSTEM      - Replace a Subsystem in a Configuration
RUNNING                - Print list of currently elaborated configurations.
SET_VERBOSITY          - Set verbosity for configuration/subsystem
SHOW_DEFAULT           - Show default configuration
SNAPSHOT               - Take a snapshot
STATISTICS             - No help available for STATISTICS
TAPE_DRIVE             - Set Tape Drive Number
TERMINAL_SETTING       - Execute terminal setting command
UNELABORATE            - Unelaborate a subsystem
VDISPLAY               - Long form display subsystem/configuration
VERBOSITY              - Print verbosity settings
```

3.2.1. ABBREVIATIONS

Print subsystem abbreviation/full-name pairs. Used to show what the short names are for corresponding subsystem names. The abbreviations can be used wherever a full subsystem name can be used. Example:

```
EEDB: abbreviations ftp
FTP_INTERFACE
UFTP
```

3.2.2. ADD_SUBSYSTEM

Add a subsystem to a configuration. Adds the subsystem(s) at the top of the configuration. Subsystem must exist, not depend on anything not already in the configuration, and may not be a subsystem already present in the configuration. Example: add initialize.6.0.0d to a mux configuration that didn't have a version of initialize.

```
EEDB: add_subsystem
Existing Configuration: mux
Subsystem.Version: init.6.0.0d
Subsystem.Version:
```

3.2.3. BUILD_CONFIGURATION

Build a configuration from another configuration. All subsystems below (and including) the parent for the new subsystem are shared between the old and new configuration. Example: build a configuration off of Mux that doesn't include Initialize, but does include the native_debugger (ND).

```
EEDB: build_configuration
New Configuration: new
Existing Configuration: mux
Parent subsystem: nd
Subsystem.Version:
```

3.2.4. COMMON

Highest common subsystem for two configurations. Used to determine if two configurations are built from each other.

```
EEDB: common
Existing Configuration: mux
Existing Configuration: new
NATIVE_DEBUGGER
```

3.2.5. COPY_CONFIGURATION

Copy a configuration. Create a complete copy of a configuration. Example: notice that the new_configuration is required; it will continue asking until it gets one. If

the new configuration exists, but isn't elaborated, it will be replaced without comment.

```
EEDB: copy_configuration
Existing Configuration: mux
New Configuration:
New Configuration: xxx
```

3.2.6. CHECK_CONSISTENCY

Check consistency of database. If it prints anything, it will be messages complaining about the internal consistency of the database.

3.2.7. DEFAULT_CONFIGURATION

Set the default configuration to be booted with EEDB is elaborated. If this is set to "Base_Configuration", EEDB will boot nothing else.

3.2.8. DELETE

Delete a subsystem or configuration. Accepts wildcards. Won't delete elaborated configurations or subsystems that are part of a configuration. This last can be used to collect garbage, e.g.:

```
EEDB: delete
Subsystem/Configuration: +.+
```

This would generate a large number of messages about subsystems that couldn't be deleted, but will delete all subsystems not mentioned in any configuration (assuming that configurations don't have '.'s in them).

3.2.9. DISPLAY

Short form display subsystem/configuration. For configurations, only the name is given. For subsystems, the name and date are displayed. Configurations will all be listed before subsystems if both are applicable.

```
EEDB: di ece.+

Subsystems :
CORE_EDITOR.6.0.0D              01/09/86 17:07:00
CORE_EDITOR.5.2.3D              01/09/86 20:33:50
CORE_EDITOR.5.2.0D              12/07/85 13:28:09
CORE_EDITOR.5.2.1D              12/14/85 14:29:15
CORE_EDITOR.6.1.0D              01/15/86 13:04:52

EEDB: di a_5+

Configurations :
A_5_7_1
A_5_8_0
```

```
EEDB: display
Subsystem/Configuration: d_9_19_0

Configurations :
D_9_19_0
```

3.2.10. ELABORATE

Elaborate a configuration. Elaborates any subsystems that have not been elaborated, but are part of the configuration. After the first configuration has been elaborated, any further configurations must have been built from the same stem as the first. For test configurations, such as OM, elaborate will do nothing if the test program has completed and has NOT been unelaborated.

3.2.11. FIND_SEGMENT

Find a segment number, indicating what subsystem it comes from.

```
EEDB: find 10513433
CORE_EDITOR.6.0.0D
```

3.2.12. HELP

Print a help message for a command.

```
EEDB: help
Help for command:
ABBREVIATIONS          - Print subsystem abbreviation/full-name pairs
ADD_SUBSYSTEM          - Add a subsystem to a configuration
BUILD_CONFIGURATION    - Build a configuration
COMMON                 - Highest common subsystem for two configurations
COPY_CONFIGURATION     - Copy a configuration
CHECK_CONSISTENCY      - Check consistency of database
DEFAULT_CONFIGURATION - Set the default configuration
DELETE                 - Delete a subsystem or configuration
DISPLAY                - Short form display subsystem/configuration
ELABORATE              - Elaborate a configuration
FIND_SEGMENT           - Find a segment number
HELP                   - Print a help message
INSERT_SUBSYSTEM       - No help available for INSERT_SUBSYSTEM
KERNEL                 - No help available for KERNEL
QUIT                   - Leave Command Interpreter
READ_TAPE              - Read from tape
RECLAIM_SPACE          - No help available for RECLAIM_SPACE
REMOVE_SUBSYSTEM       - Remove a Subsystem from a Configuration
REPLACE_SUBSYSTEM      - Replace a Subsystem in a Configuration
RUNNING                - Print list of currently elaborated configurations.
SET_VERBOSITY          - Set verbosity for configuration/subsystem
SHOW_DEFAULT           - Show default configuration
SNAPSHOT               - Take a snapshot
STATISTICS             - No help available for STATISTICS
TAPE_DRIVE             - Set Tape Drive Number
TERMINAL_SETTING       - Execute terminal setting command
UNELABORATE            - Unelaborate a subsystem
```

```
VDISPLAY              - Long form display subsystem/configuration
VERBOSITY             - Print verbosity settings
```

3.2.13. INSERT_SUBSYSTEM

The same as ADD_SUBSYSTEM, except that it allows additions in the middle of a configuration. Additional parameter, parent, required.

```
EEDB: insert
Existing Configuration: mux
Parent subsystem: init
Subsystem.Version: init.5.0.1d
subsystem INITIALIZE is already in configuration MUX
```

3.2.14. KERNEL

Starts the kernel command interpreter.

```
EEDB: kernel
Kernel:
```

3.2.15. QUIT

Leave Command Interpreter.

```
Kernel: quit
EEDB:
```

3.2.16. READ_TAPE

Read from tape.

3.2.17. RECLAIM_SPACE

Actually delete code segments not associated with any subsystem version. Since multiple version can share code segments, this involves traversing all subsystems to determine which code segments can actually deleted. This should be run after a new release and after something akin to delete +.+

3.2.18. REMOVE_SUBSYSTEM

Remove a Subsystem from a Configuration.

```
EEDB: remove_subsystem
Configuration: mux
Subsystem INITIALIZE.5.0.1D to be deleted is not unelaborated
```

3.2.19. REPLACE_SUBSYSTEM

Replace a Subsystem in a Configuration. Used to install new subsystems into a configuration. Cannot be run on an elaborated configuration.

```
EEDB: replace_subsystem
Existing Configuration: mux
Subsystem.Version: init.6.0.0d
```

3.2.20. RUNNING

Print list of currently elaborated configurations. Test configurations that have not been elaborated are marked as (partial), which means that they are built off of the running configuration, but are not currently elaborated.

```
EEDB: running
  MUX
  ED (partial)
  DT (partial)
  OM (partial)
```

3.2.21. SET_VERBOSITY

Set verbosity for configuration/subsystem when displayed using VDisplay. Operation consists of setting a particular field to be displayed or not for either subsystems or configurations (as classes, not for specific instances). Possible information is:

```
EEDB: set_verbosity
Subsystem/Configuration: subsystem
Display option: ?
Possible completions for Verbosity options
ALL_CODE_SEGMENTS          MODULE_KEY
DATE                       NAME
ELAB_CODE_SEGMENT          SUBSYSTEM_DEPENDENCIES
LIBRARY                    USER
Display option: name
True or False: true
Subsystem options are now: NAME DATE USER LIBRARY SUBSYSTEM_DEPENDENCIES
    ELAB_CODE_SEGMENT
```

3.2.22. SHOW_DEFAULT

Show default configuration, i.e. the one that will be booted when EEDB is first started. Base_Configuration is the configuration containing only EEDB, which is always elaborated.

```
EEDB: show_default
Default configuration is BASE_CONFIGURATION
```

3.2.23. SNAPSHOT

Take a snapshot

3.2.24. STATISTICS

Not implemented.

3.2.25. TAPE_DRIVE

Set Tape Drive Number. If there were more than one tape drive, would allow setting the drive to be used by the tape command.

3.2.26. TERMINAL_SETTING

Execute terminal setting command. Allows setting of:

```
    COLUMNS_PER_LINE        how many columns to use for results
    ECHO_MODE               show fields as parsed; for debugging EEDB
    LINES_PER_PAGE          how many lines on the terminal
    PAGE_MODE               should output stop when more than a page
    SETTINGS                shows the values of the settings
```

```
EEDB: term
Terminal Setting: ?
Possible completions for Terminal_Command
COLUMNS_PER_LINE              PAGE_MODE
ECHO_MODE                     SETTINGS
LINES_PER_PAGE
Terminal Setting: settings
Terminal settings: lines = 24; columns = 79
EEDB: term page
Page mode: ?
Possible completions for Boolean
FALSE   TRUE
Page mode: false
```

3.2.27. UNELABORATE

Unelaborate a subsystem.

3.2.28. VDISPLAY

Long form display subsystem/configuration. The line of dashes give information about the configuration that makes up EEDB. Subsystems with the sequence number .XXX are not registered with EEDB.

```
EEDB: vd mux

Configurations :
MUX
```

```
    INITIALIZE.5.0.1D              12/11/85 12:19:28 Key: 1AFF3C04
    NATIVE_DEBUGGER.5.1.5D         12/15/85 18:14:44 Key: 1AFD8004
    ARCHIVE.5.0.5D                 12/11/85 23:43:04 Key: 1AFD2404
    ...
    OM_MECHANISMS.5.0.2D           12/26/85 14:51:16 Key: 1ACA2804
    TEST_UTILITIES.4.0.1D          09/16/85 19:06:56 Key: 1AC9E004
    NETWORK.5.0.3D                 11/18/85 08:37:36 Key: 1AC96004
-------------------------------------------------------------------------
    ELABORATOR_DATABASE.5.0.0D     11/08/85 14:27:20 Key: 1AC80C04
    OS_UTILITIES.5.1.0D            12/06/85 13:03:07 Key: 1A83CC04
    ...
    MACHINE_INTERFACE.4.0.1        09/03/85 15:41:02 Key: 04002C04
    ADA_BASE.4.1.0                 10/14/85 12:53:40 Key: 00010004
    MICROCODE.4.92                 12/16/85 16:54:01

EEDB: vd init.5.0.1d

Subsystems :
INITIALIZE.5.0.1D              12/11/85 12:19:28 DRK
    Lib:  :NET:CURLY:PDD:INITIALIZE.5.0.1:LIBRARIES:INITIALIZE.LIB
    With: R1000_CODE_GEN                DIRECTORY
          BASIC_MANAGERS                KERNEL
          KERNEL_DEBUGGER               OS_COMMANDS
          COMMANDS                      MACHINE_INTERFACE
          INPUT_OUTPUT                  MISCELLANEOUS
          OM_MECHANISMS                 CORE_EDITOR
          ABSTRACT_TYPES                ADA_MANAGEMENT
          PARSER                        ELABORATOR_DATABASE
          ADA_BASE                      TOOLS
          ENVIRONMENT_DEBUGGER
    Elab: 1809431
    Code: 252951    231447    10763279  251927    1809431   1808407
```

3.2.29. VERBOSITY

Print verbosity settings.

```
EEDB: verbosity
Subsystem fields to be displayed when printing
Configurations :  NAME DATE USER MODULE_KEY
Subsystems :      NAME DATE USER LIBRARY SUBSYSTEM_DEPENDENCIES
    ELAB_CODE_SEGMENT
EEDB: set_verbosity
Subsystem/Configuration: configuration
Display option: ?
Possible completions for Verbosity options
ALL_CODE_SEGMENTS          MODULE_KEY
DATE                       NAME
ELAB_CODE_SEGMENT          SUBSYSTEM_DEPENDENCIES
LIBRARY                    USER
Display option: user
True or False: true
Configuration options are now: NAME DATE USER MODULE_KEY
```

4. Procedures for System Hang Condition

To perform preliminary diagnosis and provide the maximum amount of information for diagnosis of a system crashdump tape, the following procedures should be performed prior to crashing the system and taking a crashdump. If possible, Rational should be contacted prior to crashing a "hung" system. Whenever a crashdump tape is taken, it is very important to include as much information about the crashdump as possible. The following procedures describe the minimal amount of information necessary based on the type of the system hang.

No System Console Response

The system console might be flow controlled, which will make it appear as if the system is hung.

1. Does the console respond to the BREAK key with the "Please enter 0/1/2" message? If not, try typing "2", followed by a <CR>. If the console now responds to the BREAK key, then the console was flow controlled by the IOA/C waiting for the "0/1/2" response. If the console still does not respond, then there is a problem with the console terminal, the connection to the IOA/C, the IOA/C itself, or another piece of hardware. Verify that the terminal is functioning correctly (power the terminal OFF, then ON to initiate self testing). In the event that no response can be obtained from the R1000, a crashdump tape will not be useful, and Rational should be called immediately to coordinate further diagnosis of the problem.

2. Does the console switch banners in response to ^Z? If not, try typing <CR>. If there is still no response, then type ^Q. If the console now responds to ^Z or <CR>, then the console was flow controlled by R1000 or IOP/C software. If the console still does not respond, then crash the system (using the BREAK key) and generate a crashdump tape indicating in the comments that the console would only respond to the BREAK key, and that this step had been reached.[1]

System Console Responds

Having reached this point, it has been determined that the system console will respond. If the prompt on the console is "CLI>" or "Enter configuration to boot..", then the machine has crashed, and the cause of the crash was displayed in previous output to the console. If this output is still visible, examine it to determine the cause of the crash and what was recommended as appropriate action. If a crashdump tape was prompted to be taken, then take one now and enter the displayed reason for the crash.

It is important to note that:

[1] A problem probably exists in Microcode or Kernel software.

1. Both interpreters (Kernel and EEDB) can present the "Kernel:" prompt.

2. By typing <CR> and using ^Z one can cycle through the prompts to determine which interpreter is displaying the "kernel:" prompt (indicated in the banner information).

3. To get the "Kernel:" prompt via the EEDB interface, type "kernel" at the EEDB prompt.

4. It is always better to use the kernel prompt under the EEDB banner. If the EEDB becomes hung, the non-faulting Kernel interpreter is still available.

Can the Kernel or EEDB command interpreters be reached? If not, gather the following information:

1. The output produced by BREAK-2.

2. The banners displayed by ^Z.

3. The prompts that are displayed on the console (use ^Z).

then crash the system (using the BREAK key) and make a crashdump tape, including this information along with the crashdump tape.

Having reached this point, it has been determined that the R1000 is still running, and that the Kernel/EEDB interpreters respond.

1. Do "Show_Disk_Summary" at the Kernel prompt. Does it show unrecoverable disk errors? If so, call Rational. This is most likely the reason for the hang.

2. Does "Show_Disk_Summary" show IO in progress? If so, execute the command several times. If the successive displays show no change in the total read/writes columns, yet the display shows disk IO in progress, there is an IO hang. Execute a "Show_Task_States", record the information and submit it along with a crash dump tape.

3. Do "Show_Volume_Summary". If any volume shows 0 unused capacity, then the system has reached the suspend system GC threshold. This can be verified by noting that no waiters exist for the suspend threshold for that volume. Do not submit a crash dump. Reboot following the procedures in the System Manager's Guide for recovering when the Suspend_System threshold has been reached.

4. Do a "Jobs" (threshold 1) command. Do executions of this command show activity in the CPU and D/S categories? There should be either virtually no activity, or some job(s) consuming virtually all cycles.

5. If the problem is too much activity, wait 5 or 10 minutes before proceeding. If the activity is in user jobs, then disable the user job, and do not submit a crashdump. Determine what the user job is doing and take the appropriate action. If the activity is in job 4 or 5, record several executions of the "Show_Task_States" (Cache) and submit this information along with a

crashdump tape.

Having reached this point, the R1000 is still running, the Kernel/EEDB interpreter respond, and the system appears to be idle, yet it also appears to be hung.

1. Use "Show_Error_Log" and search for anything out of the ordinary.

2. Can users login? If not, try more than one connection. If telnet is being used, try a serial (DH11) port (like 16).

3. Are users logged in but getting no response? Are all users having this problem? It is possible for a user's session to become hung, which doesn't imply that all sessions are hung. If not all user's sessions are hung, examine the session error logs (in !Machine.Error_Logs) for the hung sessions. Also do a "Show_Task_States" (Cache) and see if any task is in an "Environment_Debugging_Wait". If so, execute the Show_Tasks procedure in the System_Maintenance subsystem, using the task ID of the task shown to be in the "Environment_Debugging_Wait", and call Rational with the output of this command.

Having reached this point, the most effective means of debugging the problem is to contact Rational and have a remote debugger connected to the system. A crashdump tape will generally prove inconclusive.

Failure to provide the information described above along with a crashdump tape will generally result in an undiagnosible crashdump tape. A description of simply "system hang" is not sufficient to diagnosis a system hang.

Table Of Contents

Table Of Figures