Sun StorEdge™ A7000 Online Exerciser Reference Manual

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300 Fax 650 969-9131

Part No. 805-4896-10
January 1999, Revision A

Send comments about this document to: `docfeedback@sun.com`

Please
Recycle

Contents

Figures

Tables

Preface

Sun StorEdge A7000 Online Exerciser Reference Manual is specific to the Online Exerciser provided for testing the Sun StorEdge A7000 Intelligent Storage Server System. This manual contains the following information:

- A description of the Online Exerciser file structure.
- Descriptions of the Online Exerciser user interface menu formats and the key commands used to manipulate menu entries.
- Procedures for customizing the Online Exerciser for your system configuration.
- Procedures for running and terminating Online Exerciser tests.
- Descriptions of the individual exerciser tests provided with the Online Exerciser.
- Descriptions of the Online Exerciser log file message formats.

How This Book Is Organized

Chapter 1 "Online Exerciser Overview" describes the Online Exerciser components and the file system structure.

Chapter 2 "User Interface Structure and Key Commands" describes the Online Exerciser user interface. This chapter includes sample menu displays and descriptions of keys on the console keyboard that are used to manipulate the various menus and tables.

Chapter 3 “Operating Procedures” describes the procedures used to customize the Online Exerciser for running specific tests. Procedures are provided for performing the following operations:

- Installing and executing the exerciser.
- Reconfiguring the system.
- Modifying the tests and remote systems lists.
- Modifying test groups.
- Executing test groups.
- Terminating test group execution.

Chapter 4 “Log File Message Formats” describes the different types of messages displayed on the console or logged to a file during Online Exerciser execution.

Chapter 5 “Disk Exerciser” describes the Online Exerciser tests used to verify disk drives. The following information is provided:

- Special requirements for test execution.
- Descriptions of the unique test parameters for this program.
- Test descriptions.
- Descriptions of program messages associated with this program.

Chapter 6 “Memory Exercisers” describes the Online Exerciser tests used to verify system memory. The following information is provided:

- Special requirements for test execution.
- Descriptions of the unique test parameters for memory testing.
- Test descriptions.
- Descriptions of program messages associated with memory tests.

Chapter 7 “Network Exerciser” describes the Online Exerciser tests used to test remote systems configured through Ethernet on a TCP/IP network. The following information is provided:

- Special requirements for test execution.
- Descriptions of the unique test parameters for this program.
- Test descriptions.
- Descriptions of program messages associated with this program.

Chapter 8 “Tape Exercisers” describes the Online Exerciser tests used to test tape drives configured in the system. The following information is provided:

- Special requirements for test execution.
- Descriptions of the unique test parameters for these programs.
- Test descriptions.
- Descriptions of program messages associated with these programs.

Chapter 9 “UMAX V Exerciser” describes the Online Exerciser tests used to verify the ability of the operating system to execute commands and system calls. These tests are also used to measure system performance. The following information is provided:

- Special requirements for test execution.
- Descriptions of the unique test parameters for this program.
- Test descriptions.
- Descriptions of program messages associated with this program.

Chapter 10 “Generic Exerciser” describes the Generic Exerciser provided for incorporating user defined commands and test scripts into the Online Exerciser. The following information is provided:

- A description of the unique test parameter provided with the Generic Exerciser.
- Examples of commands that could be used to test various system components.
- Descriptions of the general milestone and error messages associated with executing user defined commands and test scripts.

Appendix A “Man Pages” describes the man page information for the Online Exerciser. This information is displayed when you enter **`man oe`** at the UMAX™ V prompt.

Typographic Conventions

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output.	`specified drive is not in configuration`
`AaBbCc123`	What you type, when contrasted with on-screen computer output.	`tar` *filename*
AaBbCc123	Book titles, new words or terms, words to be emphasized. Variable expressions; replaced with a real name or value.	Read Chapter 2 in the *Sun StorEdge A7000 Online Exerciser Programmer's Guide*. `Capacity:` *x* `bytes`
`[ ]`	In system output examples, brackets indicate optional values. If several values are placed inside brackets, any or none of them can be displayed. Brackets are also used in system prompts to enclose the response choices.	In the following example, entering options is optional: **`setpgrp /usr/oe/oe`**`[`*options*`]` **`-mb`** *pathname* **`&`**
`{ }`	In program message examples, braces indicate that one of the enclosed values **must** be displayed. A vertical bar separates the values.	In the following example, either `physical` or `virtual` **must** be displayed: `Shifting {physical|virtual} address 0x`*aaaaaaaa*
`...`	The horizontal ellipsis indicates repetition or omission.	In the following example, one or more options can be entered: **`setenv OEINIT "`***opt1 opt2*`..."`

TABLE P-1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Examples
. . .	The vertical ellipsis indicates continuation of system output.	In the following example, additional information would be displayed in place of the vertical ellipsis: *rms* . . .
< >	In examples of command input, an item surrounded by a greater than and less than sign must be replaced with an action. The greater than and less than signs are omitted when performing the action.	<CR> means press the Return key.
^	A circumflex followed immediately by a letter or punctuation mark indicates a control-character sequence. Enter a control-character sequence by pressing and holding the Control key while typing the specified character.	**^C** means press and hold the Control key while typing C.

Related Documentation

TABLE P-2 Related Documentation

Type	Title
Programmers Guide	*Sun StorEdge A7000 Online Exerciser Programmer's Guide*

Sun Documentation on the Web

The `docs.sun.com` web site enables you to access Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at:

`http://docs.sun.com`

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

docfeedback@sun.com

Please include the part number of your document in the subject line of your email.

CHAPTER 1

Online Exerciser Overview

The Online Exerciser is a system exerciser that runs under the UMAX V operating system. Unlike other diagnostics used to test the hardware at power up or reset time, the Online Exerciser exercises both hardware and software under normal system operating conditions.

The Online Exerciser can be expanded to contain tests for any type of hardware configured in the Sun StorEdge A7000 Intelligent Storage Server System. It can also be used to test remote systems connected using Ethernet on a TCP/IP network.

Exerciser Components

The Online Exerciser has two components:

- The control program.
- The test library and test programs.

Control Program

The control program provides an interface between the operator and the individual tests. It collects and logs execution information including pass, milestone, and error reports from the tests as the tests are run. This information can be logged to a file, displayed on a terminal, or sent to a printer. Refer to Chapter 4 for descriptions of the different types of messages.

The control program also provides a menu-driven interface that allows you to perform the following operations:

- Modify the testing configuration.
- Modify individual test groups.
- Add or delete remote systems in the remote library.

- Add or delete tests in the local test libraries.
- Execute the tests in a test group.
- Exit the Online Exerciser.

Test Library and Test Programs

A test program is a set of functions that perform the actual testing. The test programs available for running on the system are combined into a test library.

Test programs are selected from the test library and combined to form a test group. This group may contain multiple copies of the test programs, a single copy of each program or a combination of single and multiple copies. Not all the test programs need to be in the test group. It is also possible to have more than one test group. When the Online Exerciser is started, you enter the name of the test group you want to run.

Each test in a test program is divided into subtests or cases. For example, a disk test may have a sequential seek/read case, a random seek/read case, a sequential seek/read/write case, and a random seek/read/write case. The test program runs each case individually. When all the cases have been run, one pass is completed.

Each test also has parameters used to define the testing environment. When you select parameter values for a set of tests in a test group, the parameters are saved as part of the group. Common parameters include the number of passes to run, stop on error, loop on error, ignore error, and execute or skip individual subtests.

File System Structure

Once the Online Exerciser is installed and built, all the files it needs to execute are automatically placed into the file system in the structure shown in FIGURE 1-1.

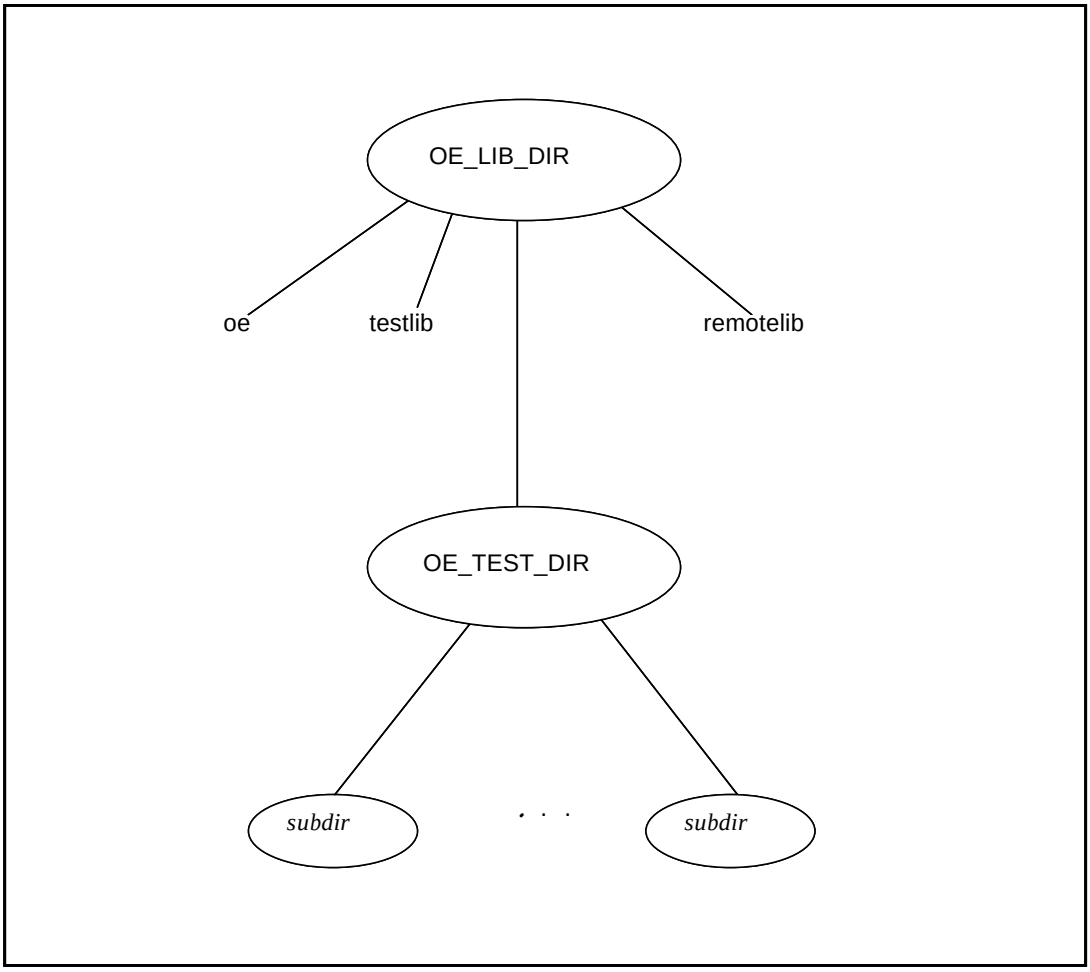


FIGURE 1-1 Online Exerciser File System

subdir specifies one of the test subdirectories.

Library Directory

The library directory (`OE_LIB_DIR`) is the parent directory for all the Online Exerciser files. The default directory is `/usr/oe`. Under this directory are programs, files containing lists of tests for local and remote systems, and subdirectories for the individual Online Exerciser tests.

Test Directory

The test directory (`OE_TEST_DIR`) contains subdirectories for the individual tests and their related files. The default directory is `test`.

This directory contains the following subdirectories:

- `devel` - a bare minimum sample test program and related files
- `disk` - disk tests and related files
- `tape` - tape tests and related files
- `network` - network tests and related files
- `umaxtest` - UMAX V operating system tests and related files
- `memory` - memory tests and related files
- `generic` - user-defined commands, test scripts and related files

Additional tests can be added by making other subdirectories with additional tests and related files.

Note – The information in the `devel` subdirectory is provided for users developing or porting test programs to be run under the control of the Online Exerciser. Refer to the *Sun StorEdge A7000 Online Exerciser Programmer's Guide* for additional information.

The test directory contains additional subdirectories that are not listed here. The tests in those subdirectories are not applicable to the Sun StorEdge A7000 Intelligent Storage Server System.

Individual Files

The following individual files are located directly under the library directory in the file system:

- `oe`
- `testlib`
- `remotelib`

oe

The oe file is an executable file containing the control process for the Online Exerciser.

testlib

The testlib file contains a list of tests available on the local host.

remotelib

The remotelib file contains a list of the remote systems available for testing.

CHAPTER 2

User Interface Structure and Key Commands

The Online Exerciser user interface is composed of menus, submenus, and prompts used to manipulate exerciser tests, test parameters, system configurations, and execution options. Various keys on the console keyboard are used to make menu selections, move through the menu structure, enter data, answer prompts, or exit the user interface.

Menu Formats

Most of the user interface menus are displayed in the same format. The top of the screen contains the menu name and a horizontal display of menu selections. The center of the display contains a multicolumn vertical table of menu selections. At the bottom of the screen a horizontal line of key commands is displayed, followed by the Online Exerciser copyright message. Not all menus use this format. Some screens do not have a linear display of menu selections, but use the table display only. Others do not use a table display. FIGURE 2-1 shows a sample menu display containing both linear and tabular menu selections:

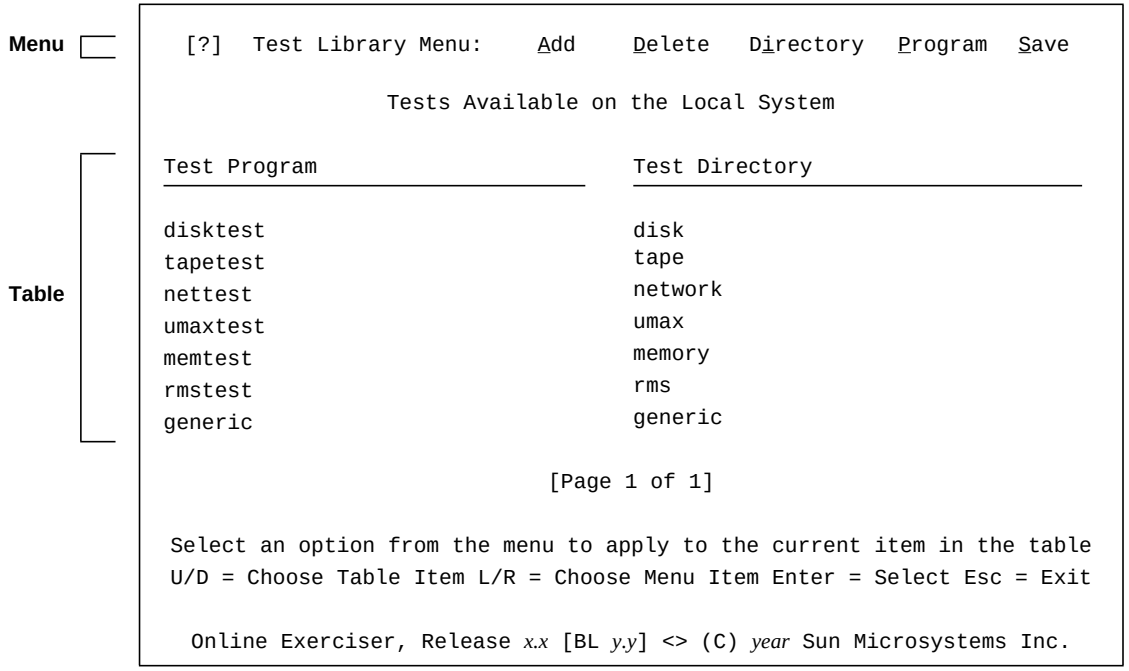


FIGURE 2-1 Sample Menu Display

Variable	Description
x.x	Specifies the current release level of the Online Exerciser.
y.y	Specifies the current baseline of the Online Exerciser.
year	Specifies the year the program was copyrighted.

Key Commands

The following sections identify the keys used when interfacing to the different menu layouts.

Standard

The following keys are used for the described functions on any type of menu display. Most of the functions can be performed in more than one way.

Key	Alternate Key	Function
Line Feed	Ctrl J	Select an item from the menu
<CR>	Ctrl M	Select an item from the menu
Escape	Ctrl G	Exit the menu

Note – If a letter in a menu selection is underlined, the underlined letter can be used as a key command shortcut.

Line Format

In addition to the standard key commands, use the following keys when the menu is displayed horizontally.

Key	Alternate Key	Function
←	Ctrl B	Move to the left
→	Ctrl F	Move to the right

The Tab key is also used to move to the next item in the menu.

Table Format

In addition to the standard key commands, use the following keys when the menu is displayed in table format:

Key	Alternate Key	Function
↓	Ctrl P	Go to the previous item
↑	Ctrl N	Go to the next item

Parameters

Parameters are areas on a display screen or menu that can be modified, not just selected. There are two different types of parameters:

- String
- Discrete

String Parameters

String parameters are provided for data entry. Use the following keys to manipulate the contents of string parameters:

Key	Function
Backspace	Deletes the character to the left of the cursor
Ctrl A	Go to the beginning of the line
Ctrl E	Go to the end of the line
Ctrl D	Delete the character under the cursor
Ctrl K	Erase from the cursor to the right
Ctrl U	Undo any editing and return to starting entry

The left and right arrow keys can also be used in string parameters. When you're working with string parameters, insert mode is always in effect.

Discrete Parameters

Discrete parameters contain a limited number of selections, with one selection visible at any time. Use the following keys to manipulate discrete parameter entries:

Key	Function
`Ctrl A`	Go to the first selection
`Ctrl E`	Go to the last selection
`Ctrl U`	Undo any editing and return to starting entry

The left and right arrow keys can be used to display another selection. The space bar can also be used to cycle through the option list.

Miscellaneous

The following keys are used to perform miscellaneous operations:

Key	Function
`Ctrl L`	Redraw the screen
`Ctrl A`	Go to the first item on the current page
`Ctrl E`	Go to the last item on the current page
`Ctrl V`	Go to the top of the next page
`Ctrl R`	Go to the top of the previous page
<	Go to the top of the first page
>	Go to the top of the last page

Note – The terminal interrupt key is used to terminate an active test group. This key is described in the Chapter 3.

CHAPTER 3

Operating Procedures

This chapter identifies the procedures used to perform the following operations:

- Start the exerciser and display the main menu.
- Reconfigure the system.
- Modify the list of available tests.
- Modify the list of available remote systems.
- Modify a test group.
- Run the tests in a test group.
- Exit the exerciser.
- Terminate an active test group.

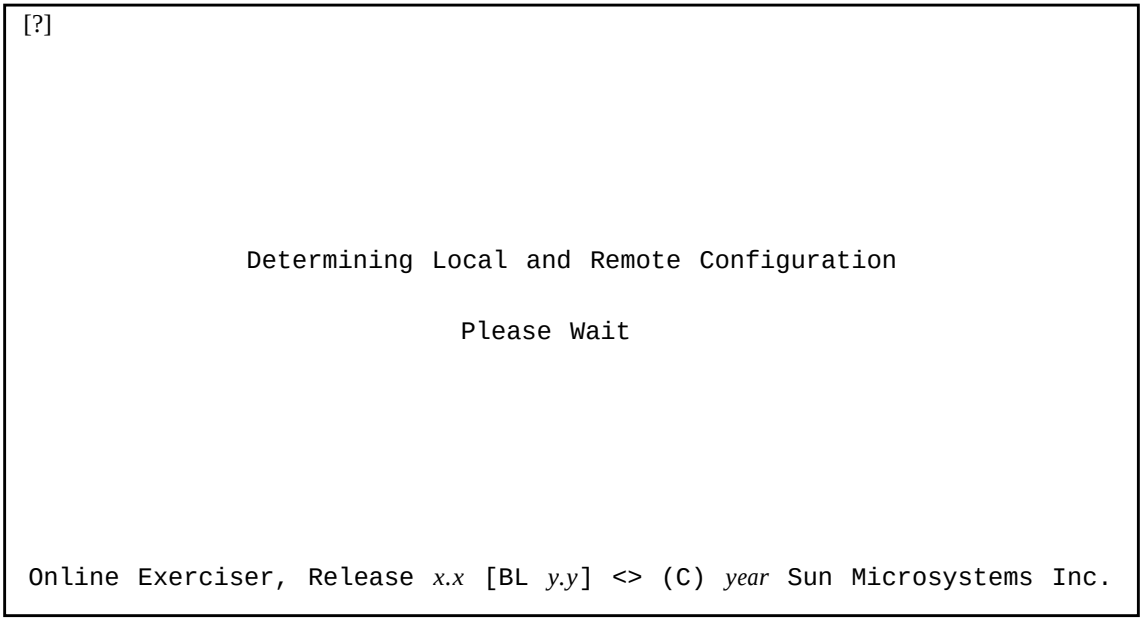
Starting the Exerciser

Perform the following operations to start the Online Exerciser:

1. Ensure that you have root access privileges by logging in as root or becoming superuser.
2. Change to the library directory and type:

```
cd /usr/oe
./oe
```

The exerciser displays the following message and begins to determine the configuration of the local and remote systems:



Variable	Description
x.x	Specifies the current release level of the Online Exerciser.
y.y	Specifies the current baseline of the Online Exerciser.
year	Specifies the year the program was copyrighted.

After establishing the system configuration, the Online Exerciser displays its main menu in the following format:

```
[?] Main Menu:   Configure  Group  Remote-Lib   Test-Lib   Execute  Quit




        Select an item from the menu at the top of the screen.
  Left/Right = Prev/Next Item Enter = Select Item Escape = Exit Menu

Online Exerciser, Release x.x [BL y.y] <> (C) year Sun Microsystems, Inc.
```

Reconfiguring the System

Reconfigure the Online Exerciser to include any changes made to the Remote or Test Libraries by selecting the `Configure` entry from the main menu. The exerciser then displays the following prompt:

```
Reconfigure all local and remote tests:  Yes     No
```

Select `No` to return to the main menu or `Yes` to change the system configuration. If you select `Yes`, the Online Exerciser checks the configuration and redisplays the main menu.

Changing the Test Library

The test library contains a list of all the tests available on the local system. Use the `Test-Lib` selection on the main menu to modify the contents of the test library by adding tests, deleting tests, or editing the current test program and directory values.

Note – After the test library is modified, the system must be reconfigured with the main menu `Configure` selection.

Once `Test-Lib` is selected, the Online Exerciser displays the contents of the test library in the following format:

```
[?]   Test Library Menu:  Add   Delete   Directory   Program   Save

                  Tests Available on the Local System

Test Program                          Test Directory

disktest                              disk
tapetst                               tape
nettest                               network
umaxtest                              umax
memtest                               memory
rmstest                               rms
generic                               generic


                                [Page 1 of 1]

Select an option from the menu to apply to the current item in the table
U/D = Choose Table Item L/R = Choose Menu Item Enter = Select Esc = Exit

Online Exerciser, Release x.x  [BL y.y] <> (C) year Sun Microsystems, Inc.
```

The Test Library menu combines both table and menu selections. Use the key commands to highlight a test program and directory in the table, and then select the desired operation from the menu.

Adding a Test

The Add selection opens a new line between the current (highlighted) line and the next line. Use this selection to add new test programs or test directories to the library. After selecting Add, choose Directory to add the directory name and Program to add the test program name. When you select Directory, the exerciser displays the following string parameter:

```
Test Directory Name:
```

Enter the name of the directory in the OE_TEST_DIR directory containing the new test program.

Selecting Program from the menu displays the following string parameter:

```
Test Program Name:
```

Enter the name of the new test program.

Deleting a Test

The Delete selection removes the current (highlighted) entry from the Test Library. After you select Delete, the exerciser requests confirmation with the following prompt:

```
Delete test 'testname' from test library:  Yes     No
```

Select Yes to delete the specified test program (*testname*) from the library or No to cancel the operation.

Editing a Test Library Entry

To change the name of the test program or test directory in a specific test library entry, perform the following steps:

- Highlight the desired entry.
- Select Program (to modify the test program name) or Directory (to modify the test directory name) from the menu.
- Modify the information displayed in the string parameter.

Saving Test Library Changes

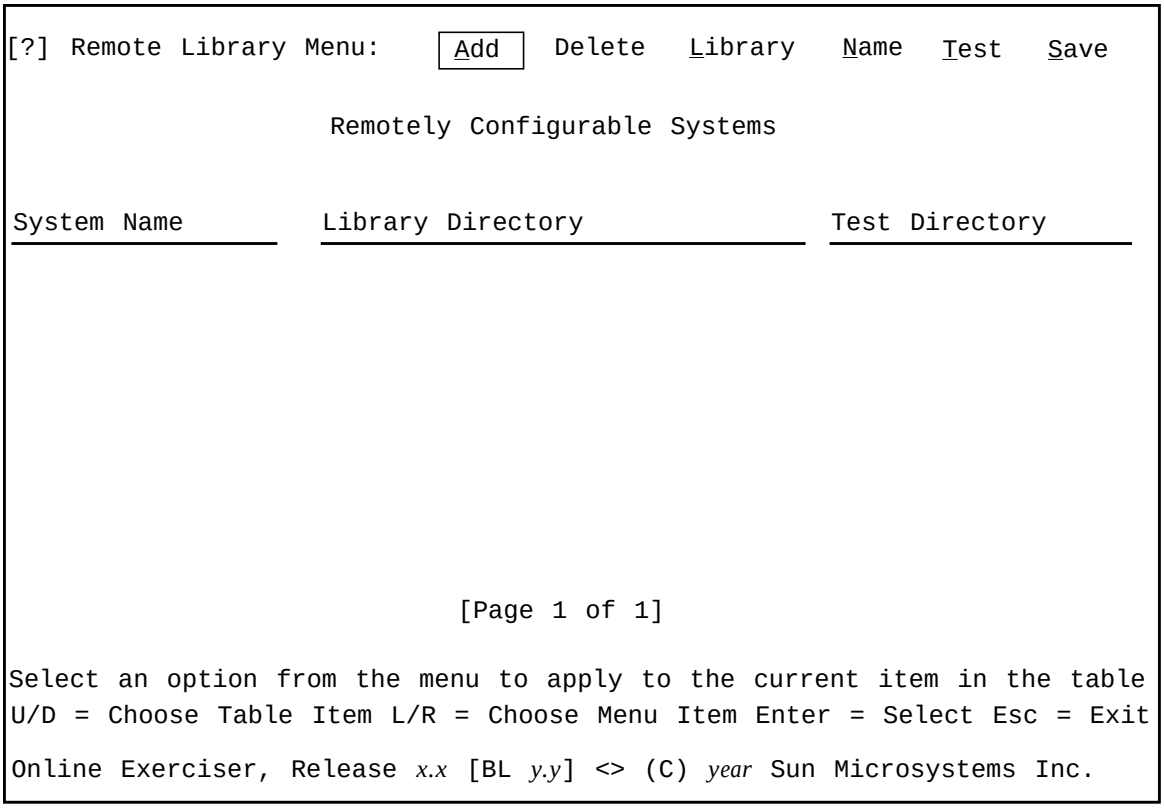
After all desired changes have been made to the test library, use the `Save` selection to save the changes to the library directory. The exerciser requests confirmation with a prompt. Select `Yes` to write the changes to the existing test library or `No` to abort the operation.

Changing the Remote Library

The remote library contains a list of the remote systems available for testing and the location of the test and library directory on each system. Use the `Remote-Lib` selection on the main menu to add or remove a remote system from the list or modify the locations of the test and library directories.

Note – After the remote library is modified, the system must be reconfigured with the main menu `Configure` selection.

Once `Remote-Lib` is selected, the Online Exerciser displays the contents of the remote library in the following format:



The Remote Library menu combines both table and menu selections. Use the key commands to highlight an entry in the table, and then select the desired operation from the menu.

Note – The system to be tested remotely must be declared in the `/etc/host.equiv` file and the `/.rhosts` file.

Adding an Entry

To add a remote system to the remote library, select Add from the menu. A blank line appears between the current (highlighted) line and the next line. Select Name to add a new system name. The exerciser displays the following string parameter:

```
Remote System Name:
```

Enter the name of the new system, and then select Library to display the following string parameter:

```
Remote Library Directory:
```

Enter the name of the Online Exerciser library directory on the remote system. The library directory name is typically /usr/oe. Finally, select Test to display the following string parameter:

```
Remote Test Directory:
```

Enter the name of the Online Exerciser test directory on the remote system. The test directory name is typically test.

Deleting an Entry

To delete a remote system from the remote library, highlight the entry for the specified system and select Delete to remove it from the library. The exerciser requests confirmation with the following prompt:

```
Delete system 'sysname'  from remote library:  Yes  [No]
```

Select Yes to delete the specified system (*sysname*) or No to abort the operation.

Editing an Entry

To change a specific remote library entry, perform the following steps:

- Highlight the desired entry.
- Select `Name` (to modify the remote system name), `Library` (to modify the remote library directory), or `Test` (to modify the remote test directory) from the menu.
- Modify the information displayed in the string parameter.

Saving Remote Library Changes

After all desired changes have been made to the remote library, use the `Save` selection to save the changes to the remote library directory. The exerciser requests confirmation with a prompt. Select `Yes` to write the changes to the existing remote library or `No` to abort the operation.

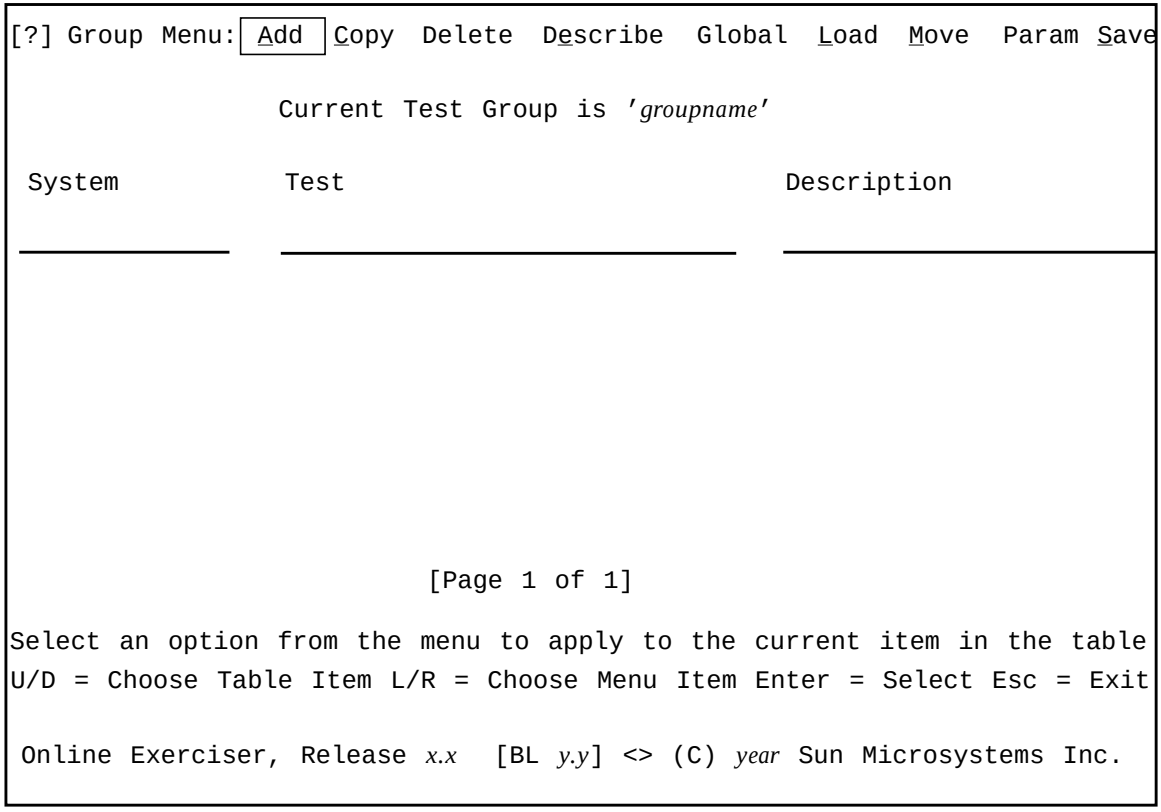
Modifying a Test Group

The test programs run by the Online Exerciser are combined into test groups. The exerciser runs the test programs either serially or in parallel depending on the global parameter values.

Use the `Group` selection on the main menu to perform the following operations:

- Add test programs to a test group.
- Delete tests from a test group.
- Modify test descriptions.
- Modify global and individual test parameters.
- Move tests in a group.
- Load a test group from a file.
- Save a test group to a file.

Once `Group` is selected from the main menu, the Online Exerciser displays the Group menu in the following format.



Variable	Description
groupname	Specifies the name of the current test group. If no test group exists, the group name is `Untitled.`

Loading a Test Group

The `Load` selection on the group menu is used to load an existing test group into the exerciser for modification. To load a test group, select `Load` and enter the name of the test group at the prompt or press the Return key to load the current test group. The Online Exerciser remembers the last test group modified. The exerciser automatically loads the test group and displays the available tests. At this time, use the other Group menu options to modify the test group.

Adding a Test

To add a test to the test group, select `Add` from the Group menu. The exerciser displays a list of the available test programs in the following format:

```
[?]  Select a test from the configuration to add to the current group:


System      Test                        Directory      Binary
--------    ------------------------    -----------    -------------
headroom    Disk Exerciser              disk           disktest
headroom    Tape Exerciser              tape           tapetst
headroom    Network Exerciser           network        nettest
headroom    UMAX V Test Suite           umax           umaxtest
headroom    Memory Exerciser            memory         memtest
headroom    RMS Exerciser               rms            rmstest
   .            .                          .              .
   .            .                          .              .
   .            .                          .              .

                             [Page 1 of 1]

                 Select an item from the table above.
 Up/Down = Prev/Next Item   Enter = Select Item Escape = Exit Table

Online Exerciser, Release x.x  [BL  y.y] <> (C) year Sun Microsystems Inc.
```

The exerciser displays both local and remote tests in the available tests table. Remote tests are displayed only if the system was declared in `/etc/host.equiv` and `/.rhosts`. `System` specifies the system containing the test, `Test` specifies the test name, `Directory` specifies the directory containing the test, and `Binary` specifies the binary test program name. Highlight the entry to be added to the test group and press the return key. The exerciser redisplays the Group menu with the new test added to the list.

Copying a Test

The Copy selection on the Group menu is used to create multiple copies of a test in the test group. To use this feature, there must be tests in the test group. To make a copy of a test, perform the following operations:

- Highlight the test to be copied.
- Select Copy from the menu and press the Return key.

The exerciser displays the following message at the top of the test group table:

```
Select position for copy of name(description).
```

Variable	Description
name	Specifies the name of the test to be copied.
description	Specifies a brief description of the test.

Select a position in the test group table for the test copy and press the Return key. The copy is inserted between the current (highlighted) line and the next line. Repeat the procedure to create multiple copies of a test.

Deleting a Test

Use the Delete selection on the Group menu to delete a test from the test group. Highlight the test to be removed and select Delete. The exerciser requests confirmation with the following prompt:

```
Delete This Group Entry:  Yes  [No]
```

Select Yes to delete the highlighted entry or No to abort the operation.

Changing a Test Description

The `Describe` selection in the group menu is used to provide a test description for each test in the group. This selection can also be used to modify an existing test description. To add or change a test description, perform the following operations:

- Highlight the desired entry.
- Select `Describe` from the menu.
- Modify the string parameter displayed by the exerciser to contain the desired test description.

Moving a Test

The `Move` selection on the Group menu is used to move a test from its current location in the test group to another location. Since the exerciser runs the tests in sequential order, use `Move` to change the execution order. To move a test, perform the following operations:

- Highlight the test to be moved.
- Select `Move` from the menu and press the Return key.

The exerciser displays the following message at the top of the test group table:

```
Select new position for name(description)
```

Variable	Description
name	Specifies the name of the test to be moved.
description	Specifies a brief description of the test.

Select a position in the test group table and press the Return key. The test is moved from its current location and placed between the current (highlighted) line and the next line.

Modifying Global Parameters

Global parameters are used to influence the execution of all the tests in a test group. When `Global` is selected from the Group menu, the exerciser displays the global parameters for the current test group. The following shows the default parameter values:

```
[?]
            Global Parameters for Test Group 'groupname'


                  Run Mode: [Parallel]

                  Log Errors : Yes
                  Log Milestones : Yes
                  Log Cases      :  Yes
                  Log Passes     : Yes


                  Log Filename:


                  Logged Pass Frequency:



          Enter the string and discrete values in the form
     Up/Down = Prev/Next Item  Enter = Accept values Esc = Abort

  Online Exerciser, Release x.x [BL y.y] <> (C) year Sun Microsystems Inc.
```

Variable	Description
groupname	Specifies the name of the test group associated with these parameters.

Run Mode Parameter

The run mode parameter defaults to running the tests in parallel mode. The tests are started and run concurrently. In serial mode, each test is run sequentially through all passes before the next test in the test group is started.

Log Option Parameters

The log option parameters are used to filter undesired messages from the message log. The parameters default to logging all messages. To filter a specific type of message out of the log, change the parameter value to `No`. You can use the space bar or the right and left arrow keys to cycle through the options for discrete parameters. The log filename is the name of the file where messages will be logged. If the filename is a blank space or a dash (-), messages are logged to the standard output. The log file may be any valid filename or device name.

The logged pass frequency is used to specify how frequently messages should be logged.

The global parameters you selected are saved with the test group.

Modifying Individual Parameters

In addition to the global parameters associated with the test group, each test program has its own individual parameters. Access the individual test parameters by selecting `Param` from the Group menu after adding the test to the group. The following parameters are standard for all tests:

Parameter	Default Value
Number of Passes	0
Maximum Number of Errors	0
Repeat Test	No
Debug Statements	No Debug Statements
Milestone Statements	Only Mandatory Milestones

The first two are string parameters and the last three are discrete parameters.

Each test also has a set of individual parameters used to execute or skip individual subtests.

Parameter Descriptions

Parameter	Description
Number of Passes	This parameter is used to select the number of passes executed by the test. If the number of passes is `0` and Repeat Test is `Yes`, the test will loop indefinitely. If Repeat Test is `No`, the Number of Passes value is ignored and only one pass is executed.
Number of Errors	This parameter is used to limit the number of error messages reported by the test. The test aborts when the number of errors specified is detected. If the number of errors is `0`, there is no limit to the number of errors that may occur.
Repeat Test	This parameter is the enable flag for the Number of Passes parameter. If Repeat Test is `No`, only one pass will be run. If repeat test is `Yes`, the test runs the number of passes specified in the `Number of Passes` parameter.

Debug Statements	The debug statements are messages providing test-specific information. There are four levels of output:
	`No Debug Statements`
	`All Debug Statements`
	`Major Debug Statements`
	`Most Debug Statements`

Milestone Statements	Milestone statements provide information for status and performance measurements. There are four levels of output:
	`Only Mandatory Milestones`
	`Mandatory and Major`
	`Most Milestones`
	`All Milestones`

Execute *subtest*	This parameter is used to specify executing a specific subtest in each pass of the test. There are two selections: `Execute` and `Skip`. `Skip` causes the subtest to be ignored for each pass of the test. Some subtests default to `Execute` and others default to `Skip`.

To modify the parameters for a specific test, perform the following operations:

- Highlight the desired test in the table.
- Select `Param` from the Group menu.

The Online Exerciser displays a menu of parameter values in the following format:

```
[?]  Param Value Menu:     Default     Default-All     Edit

                Parameter Values for testname (description)

Parameter Name                          Parameter Value

Number of Passes                        0
Maximum Number of Errors                0
Repeat Test                             No
Debug Statements                        No Debug Statements
Milestone Statements                    Only Mandatory Milestones

                            [Page 1 of 1]

Select an option from the menu to apply to the current item in the table
U/D = Choose Table Item L/R = Choose Menu Item Enter = Select Esc = Exit

  Online Exerciser, Release x.x [BL y.y] <> (C) year Sun Microsystems Inc.
```

Variable	Description
testname	Specifies the name of the test associated with these parameters.
description	Specifies the description associated with this test.

To change all the parameters to their default values, select `Default-All` from the parameter menu. The exerciser requests confirmation with the following prompt:

```
  Set all parameters to default values:  Yes  No
```

Select `Yes` to modify all the parameters or `No` to abort the operation and return to the Param Value menu.

To change an individual parameter to its default value, select Default from the parameter menu. The exerciser requests confirmation with the following prompt:

```
Set 'param' to 'value' :   Yes  [No]
```

Variable	Description
param	Specifies the name of the parameter.
value	Specifies the default parameter value.

Select Yes to change the parameter to its default value or No to abort the operation and redisplay the parameter menu.

To set a parameter to a value other than the default value, highlight the parameter to be modified and select Edit. The exerciser displays the parameter and current parameter value in the following format:

```
Enter value for 'param' :
```

Variable	Description
param	Specifies the name of the parameter.
value	Specifies the current parameter value.

If this is a discrete parameter, use the control or arrow keys to display the various parameter values until the desired value is shown. If this is a string parameter, enter the new parameter value. Use the Escape key to return to the Group menu.

Saving a Test Group

After building or modifying a test group, save the changes to a file with the Group menu Save selection. Once Save is selected, the exerciser prompts for the test group filename as follows:

```
Save Test Group Filename :
```

The filename entered in response to the prompt becomes the test group name. If an attempt is made to exit the Group menu without saving the test group, the exerciser displays the following prompt:

```
Current group has been modified and not saved. Confirm Exit:
Yes  No
```

Enter Yes to exit without saving the changes or No to return to the Group menu.

If the selected filename already exists, the exerciser displays the following prompt:

```
File 'filename' already exists. Confirm Save : Yes  No
```

Enter Yes to confirm writing over the existing file. Enter No to redisplay the test group filename prompt.

If the exerciser is unable to save the test group to the specified file, the following message is displayed and the test group filename prompt is redisplayed:

```
Unable to write group to file 'filename'
```

Variable	Description
filename	Specifies the name of the file selected for the test group.

Note – If you exit the Group menu without using the Save option, the current test group is cleared.

Running Exerciser Tests

Once a test group has been created and saved to a file, the tests can be run from either the main menu or a command line. The command line format allows the execution of multiple test groups. The main menu selection is used to execute a single test group.

From the Main Menu

To run the tests from the main menu, select the `Execute` option. This option allows the execution of a single test group. The Online Exerciser displays the following string parameter:

```
Test Group to Run:
```

Enter the name of the desired test group and press the Return key. To run the tests in the default test group displayed in the parameter, simply press the Return key. The Online Exerciser remembers the last test group executed. After all the tests run, the exerciser displays the following message:

```
** Run of 'groupname' complete.
** Press Enter to continue -->
```

Variable	Description
groupname	Specifies the name of the test group executed.

Press the Return key to return to the main menu display.

From a Command Line

The Online Exerciser control program provides the ability to run multiple test groups from a single command line. The command line also supports a variety of Online Exerciser control options.

To use the command line feature, exit from the user interface by selecting `Quit` on the main menu. The Online Exerciser requests confirmation with the following prompt:

```
Exit the Online Exerciser. Are you sure:  Yes  No
```

Select `Yes` to exit the Online Exerciser.

After quitting the user interface, enter a command line in the following format:

```
oe  [group ...] [opt ...]
```

Variable	Description
group	Specifies the name of the test group or groups to be executed. Tests are only executed if `-mb` is one of the selected options.
opt	Specifies an Online Exerciser command line option.

If more than one test group is specified, the Online Exerciser runs the tests sequentially in the order they are listed in the command line. Tests are not executed unless the `-mb` option is specified in the command line.

Command Line Options

One or more of the following options may be entered as part of the command line:

Option	Description
`-c`	If this option is specified, the Online Exerciser determines and displays the current configuration of all the tests on the local and remote systems. The exerciser does not run any test group or enter the group editor.
`-d`	This option enables debugging output to `stderr` and sends debugging messages to the log file if test groups are run.
`-l` *arg*	This option allows overriding the library directory (`OE_LIB_DIR`). The default library directory is `/usr/oe`. Enter the complete path name (*arg*) of the new library directory.
`-m` *arg*	This option is used to select the Online Exerciser operation mode. The following are valid arguments: `b` Batch Mode - this argument allows test groups to be executed from the command line. `e` Curses Noninteractive Mode - this is the default mode.

Option	Description
`-o` *arg*	This option allows overriding the global log files specified in any of the test groups. The log output is sent to the file specified by *arg*. If *arg* is a **-**, the log output is directed to the standard output of the Online Exerciser.
`-q` *arg*	This option allows overriding the test directory (`OE_TEST_DIR`). The default test directory is `test`. Enter the name of the directory in the library directory where the test program subdirectories are located.
`-r` *arg*	This option allows reading the remote library from a file other than `remotelib`. The only valid argument is a new file name. The file must be in the library directory.
`-t` *arg*	This option allows reading the test library from a file other than `testlib`. The only valid argument is a new file name. The file must be in the library directory.
`-v`	This option is used to display information about the current version of the Online Exerciser.

If an option requires an argument, no space is required between the option and the argument.

Using the command line method requires entering the options as part of the command line each time you want to modify them. The options selected are valid only for the operations performed by the individual command line.

Another method of setting options uses the `setenv` command. Options selected with this command remain in effect for all command lines until you exit the operating system. To modify the options, enter a command line in one of the following formats:

From C Shell

```
setenv OEINIT "opt1 opt2 ..."
```

From Bourne/Korn Shell

```
OEINIT="opt1 opt2 ..."; export OEINIT
```

To reset the command line options, enter one of the following command lines:

From C Shell

```
unsetenv OEINIT
```

From Bourne/Korn Shell

```
unset OEINIT
```

opt	Specifies the Online Exerciser command line option.
OEINIT	Specifies the Online Exerciser environment variable.

Running in the Background

Enter the following command line to run the test group in the background:

```
setpgrp /usr/oe/oe [options] -mb pathname &
```

options	Specifies the command line options described on the preceding pages.
pathname	Specifies the complete pathname of the test group to execute.

Exiting the User Interface

To exit the user interface and return to the operating system, select `Quit` from the main menu. The Online Exerciser requests confirmation with the following prompt:

```
Exit the Online Exerciser. Are you sure:  Yes  No
```

Select `Yes` to exit the Online Exerciser or `No` to remain in the main menu.

Terminating an Active Test Group

To terminate the execution of an active test group, press the terminal interrupt key on your keyboard. This causes the Online Exerciser program to send a termination signal to each active process. As each process is notified, the Online Exerciser posts a `** Terminating **` milestone message. When a process receives the termination signal, it posts an abort message followed by a completion message.

Note – The terminal interrupt key defaults to the Delete key.

If you press the terminal interrupt key again before the termination process is complete, a `Fatal Communications Error` may be posted. This is caused by the termination of the Online Exerciser program before it could terminate the tests in the test group. In this case, the individual tests will eventually terminate but no milestone, abort, or completion messages will be posted.

The termination time is determined by the number of tests in the test group.

If the Online Exerciser is running in the background, the terminal interrupt key will not terminate execution. In this case, enter one of the following command lines to terminate test group execution:

`kill -INT` *proc_id*
`kill -2` *proc_id*

*proc_id s*pecifies the process identification number of the main Online Exerciser program.

CHAPTER 4

Log File Message Formats

Any messages produced by tests during test group execution are logged to a file or a device except those filtered with the global parameter settings. Refer to Chapter 3 for descriptions of the global parameter settings.

Message Format

Each message recorded in the log is displayed in the following format:

type system test ID date time description

Variable	Description
type	Specifies the type of message as follows: `S` Test Started `F` Test Finished `C` Case Completed `P` Pass Completed `E` Error `M` Milestone `D` Debug `A` Aborted
system	Specifies the name of the system where the test is running.
test	Specifies the binary name of the test under execution.

Variable	Description
ID	Specifies a unique identification number given to each test. Each time a test is started, a new number is assigned to identify the test to the control program. The identification number also associates a running test with its log messages.
date	Specifies the date when the message was received. The date is expressed in Month / Day / Year format.
time	Specifies the time the message was received. The time is expressed in Hours:Minutes:Seconds format.
description	Specifies a description of the action generating a message.

Message Descriptions

The following sections identify the contents of the *description* field for each type of message.

Start (S) Message

The Start message contains the following information in the *description* field:

testname (*test description*) `Started`

Variable	Description
testname	Specifies the name of the test.
test description	Specifies a brief description of the test. The test description is supplied by the user during the creation of the test group.

Finish (F) Message

The Finish message contains the following information in the *description* field:

```
testname (test description) Completed
```

Variable	Description
testname	Specifies the name of the test.
test description	Specifies a brief description of the test. The test description is supplied by the user during the creation of the test group.

Case Complete (C) Message

The Case Complete message contains the following information in the *description* field:

```
Pp Cn casename Complete
```

Variable	Description
p	Specifies the pass number.
n	Specifies the case number.
casename	Specifies the name of the completed case.

Pass Complete (P) Message

The Pass Complete message contains the following information in the *description* field:

```
Pp Pass Complete
```

Variable	Description
p	Specifies the pass number.

Error (E) Message

The Error message contains the following information in the *description* field:

```
Pp Cn error message
```

Variable	Description
p	Specifies the pass number.
n	Specifies the case number.

The actual *error message* contains test specific information.

Milestone (M) Message

The Milestone message contains the following information in the *description* field:

```
Pp Cn milestone message
```

Variable	Description
p	Specifies the pass number.
n	Specifies the case number.

The actual *milestone message* is test specific and defined by the test process. Milestone messages are used to report successes, performance measurements, or other non-error information.

Debug (D) Message

If a Debug message is displayed or logged, the *description* field contains the following information:

```
debug message
```

A Debug message provides information about the current internal operations of the test process or the Online Exerciser in conjunction with a specific test. This includes information about starting a test successfully or losing the connection to a test.

Aborted (A) Message

If an Aborted message is displayed or logged, the description field contains the following information:

reason for the abort

An Abort message occurs when a test process dies before normal termination or encounters a fatal error and cannot continue.

CHAPTER 5

Disk Exerciser

The Disk Exerciser tests all the partitions and file systems mounted on the disk drive under test. It also performs random seeks and reads to verify access to raw devices.

Special Requirements

This exerciser has the following special requirements:

- The file systems on the selected disk drive must be mounted prior to the file system tests being executed.
- The file system tests can be run in parallel if enough space is on each mounted file system to create several-megabyte temporary work files.

Unique Parameters

In addition to the parameters described in Chapter 3, the Disk Exerciser provides the following unique test parameters.

TABLE 5-1 Disk Exerciser Parameters

Parameter	Description
Disk Drive	Enter the path to the drive under test in the following format: **`/dev/rdsk/#{s|d}2`** **#** specifies a disk drive number from 0-7. **s** specifies a CPU SCSI channel.

TABLE 5-1 Disk Exerciser Parameters *(Continued)*

Parameter	Description
	d specifies a VME SCSI channel.
Block Size	Enter the block size, in bytes, for read / write transfers.
File Size	Enter the number of blocks in a test file.
Iterations	Enter the number of cycles per subtest.

Test Descriptions

The Disk Exerciser is divided into the following tests or cases:

- Raw random seeks and reads
- File system sequential writes and reads
- File system random writes and reads

Prior to executing the individual tests, the Disk Exerciser performs the following operations:

- Obtains the drive configuration and generates a list of disk devices.
- Searches the disk list for the specified disk drive.
- Opens the device to obtain the layout and capacity information. If these operations are successful, the program displays the following milestone messages:

```
Device: /dev/rdsk/drive2
Capacity: x  bytes
```

Variable	Description
drive	Specifies the selected disk drive number and channel.
x	Specifies the disk capacity in bytes.

- Reads the mount table.
- Searches for any mounted file systems and unlinks any existing temp files.

The following sections describe the operations performed by the Disk Exerciser tests.

Raw Random Seeks and Reads

This test is used to verify seek and read operations on raw devices. The test performs the following operations:

- Opens the raw disk.
- Allocates a work buffer.
- Seeds the randomizer.
- Begins executing the test loop and displays the following debug messages:

```
Starting Iteration n
Starting block x
```

Variable	Description
n	Specifies the iteration count.
x	Specifies specifies the block number.

While inside the test loop, the Disk Exerciser performs the following operations:

- Seeks to a random location. Before the seek is performed, the following milestone is reported:

```
Seeking to offset x
```

Variable	Description
x	Specifies the offset position for the seek.

- Reads a file size block of data from the seek location. The following debug message is displayed prior to the read operation:

```
Reading n bytes
```

Variable	Description
n	Specifies the number of bytes in the block of data.

- Verifies the correct block size was transferred.

The test loop is repeated the number of times specified in the Iterations parameter times the Filesize.

File System Sequential Writes and Reads

This test verifies the ability to write data into each mounted file system on the device. The test performs the following operations:

- Reads the mount table and displays the following debug and milestone messages:

```
File system name mounted on disk
Testing file systems on 'device*'
```

Variable	Description
name	Specifies the name of the file system.
disk	Specifies the location of the file system.
device	Specifies the path name of the disk drive under test.

- Allocates a work buffer

The test then locates all the mounted file systems for the device under test and performs the following operations for each file system:

- Displays the following milestone message to identify the file system under test:

```
Testing file system 'fs'
```

Variable	Description
fs	Specifies the name of the file system under test.

- Creates a temporary file in this file system.

The test then executes the following test loop:

- Generates a new randomizer seed.
- Seeds the randomizer.
- Writes Filesize blocks of data into the file. The number of bytes in each block is determined by the Block Size parameter. Each block receives unique data.
- Reseeds the randomizer.
- Reads each block and verifies that the data read equals the data written.

The testing is repeated for each file system on the disk. The number of repetitions for each file system is determined by the Iterations parameter.

File System Random Writes and Reads

This test verifies the ability to write data randomly into files on each file system on the device. The test performs the following operations:

- Reads the mount table and displays the following milestone:

```
Testing file systems on 'device*'
```

Variable	Description
device	Specifies the path name of the disk drive under test.

- Allocates a work buffer

The test then locates all the mounted file systems for the device under test and performs the following operations for each file system:

- Displays the following milestone message to identify the file system under test:

```
Testing file system 'fs'
```

Variable	Description
fs	Specifies the name of the file system under test.

- Creates a temporary file in this file system.

The test then executes the following test loop:

- Writes Filesize blocks of data at random locations in the file. Calculates the checksum for each block. The number of bytes in each block is determined by the Block Size parameter.
- Reads the data written and uses the calculated checksum to verify that the data read equals the data written.

The testing is repeated for each file system. The number of repetitions for each file system is determined by the Iterations parameter.

Error Messages

If an error occurs during Disk Exerciser operation, one of the following types of errors is reported:

- Initialization errors
- Raw Random Test errors
- File System Sequential Test errors
- File System Random Test errors

Initialization Error Messages

If an error is detected during the initialization of the device under test, the Disk Exerciser reports an error with one of the following messages:

- If the Disk Exerciser is unable to obtain the configuration of disk drives:

```
popen(Drive_List): message (Error n)
Unable to get drive configuration
```

- If the program cannot generate a list of disk devices:

```
popen(Drive_List): message (Error n)
Unable to generate list of disk devices
```

- If the disk drive specified for testing is not in the configuration list:

```
Specified drive is not in configuration
```

- If the program is unable to open the specified device to read the drive capacity:

```
open(drive, O_RDONLY): message (Error n)
Unable to open drive to read capacity
```

- If the Disk Exerciser is unable to read the drive capacity:

```
ioctl(MDIOC_RDLAY): message (Error n)
Unable to read drive capacity
```

Variable	Description
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.
n	Specifies the error number associated with a system error message.
drive	Specifies the name of the disk drive under test.

- If the Disk Exerciser is unable to open the mount table:

```
Unable to read mount table
```

Raw Random Test Error Messages

If the Disk Exerciser is unable to open the specified device for testing during the Raw Random Seeks and Reads Test, it displays:

```
open(drive, O_RDONLY): message (Error n)
Unable to open device for testing
```

Variable	Description
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.
n	Specifies the error number associated with a system error message.
drive	Specifies the name of the disk drive under test.

If a seek error occurs during raw device testing, the program displays:

```
lseek(fd, pos, 0): message (Error n)
```

Variable	Description
fd	Specifies the device under test.
pos	Specifies the seek position.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.

If a seek is performed to an incorrect location during raw device testing, the program displays:

```
Seek Error: Expected = epos, Actual = apos
```

Variable	Description
epos	Specifies the expected seek position.
apos	Specifies the actual seek position.

If the test is unable to read data from the disk, the program displays:

```
read(fd, 0xaaaaaaaa, size): message (Error n)
```

Variable	Description
fd	Specifies the device under test.
aaaaaaaa	Specifies the data buffer location.
size	Specifies the block size.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.

If a read fails because the wrong number of bytes was read from the disk, the program displays:

```
Read Error: Expected =  ee bytes, Actual =  aa bytes
```

Variable	Description
ee	Specifies the expected read transfer count (block size).
aa	Specifies the actual read transfer count (block size).

If any errors were detected during this test, the following message is displayed in addition to any other error messages:

```
Raw Random Access Test Failed
```

File System Sequential Test Error Messages

If the test is unable to open the mount table, the program displays:

```
open(/etc/mnttab, O_RDONLY): message (Error n)
Unable to read mount table
```

Variable	Description
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.

If the test is unable to read in entries from the mount table, the program displays:

```
read(fd, 0xaaaaaaaa, size): message (Error n)
```

Variable	Description
fd	specifies the name of the file being read.
aaaaaaaa	specifies the data buffer location.
size	specifies the size of the mnttab structure.
message	specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	specifies the error number associated with a system error message.

If the program is unable to create a temporary work file in the file system under test, it displays:

```
open(name, O_RDWR|O_CREAT|O_EXCL, 0644): message (Error n)
Unable to create work file
```

Variable	Description
name	Specifies the name of the work file.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.

If a seek error occurs, the program displays:

```
lseek(fd, pos, 0): message (Error n)
Unable to seek to block x
```

Variable	Description
fd	Specifies the device under test.
pos	Specifies the seek position.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.
x	Specifies the block number.

If an error occurs while writing to a temporary file during file system testing, the program displays:

```
write(fd, buf, size): message (Error n)
Unable to write block x
```

Variable	Description
fd	Specifies the device under test.
buf	Specifies the data buffer location.
size	Specifies the size of the block.

Variable	Description
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.
x	Specifies the block number.

If a read error occurs during this test, the program displays:

```
read(fd, buf, size): message (Error n)
Unable to read block x
```

Variable	Description
fd	Specifies the device under test.
buf	Specifies the data buffer location.
size	Specifies the size of the block.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.
x	Specifies the block number.

If the data read is not equal to the data written, the program displays:

```
Pos: p, Expected: 0xexp, Actual: 0xact
```

Variable	Description
p	Specifies the failing location.
exp	Specifies the data expected in the location.
act	Specifies the actual data read from the location.

After testing is completed, the Disk Exerciser performs some cleanup operations. If the program is unable to unlink the temporary file, it displays:

```
unlink(file): message (Error n)
```

Variable	Description
file	Specifies the name of the temp file.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the UMAX V Programmer's Reference manual
n	Specifies the error number associated with a system error message

If any errors were detected during this test, the program displays the following message together with the other error messages:

```
Sequential File Test Failed
```

File System Random Test Error Messages

If the test is unable to open the mount table, the program displays:

```
open(/etc/mnttab, O_RDONLY): message (Error n)
Unable to read mount table
```

Variable	Description
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the UMAX V Programmer's Reference manual
n	Specifies the error number associated with a system error message

If the test is unable to read in entries from the mount table, the program displays:

```
read(fd, 0xaaaaaaaa, size): message (Error n)
```

Variable	Description
fd	Specifies the name of the file being read.
aaaaaaaa	Specifies the data buffer location.

Variable	Description
size	Specifies the size of the `mnttab` structure.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the UMAX V Programmer's Reference manual
n	Specifies the error number associated with a system error message

If the program is unable to create a temporary work file in the file system under test, it displays:

```
open(name, O_RDWR|O_CREAT|O_EXCL, 0644): message (Error n)
Unable to create work file
```

Variable	Description
name	Specifies the name of the work file.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the UMAX V Programmer's Reference manual
n	Specifies the error number associated with a system error message

If a seek error occurs, the program displays:

```
lseek(fd, pos, 0): message (Error n)
Unable to seek to block x
```

Variable	Description
fd	Specifies the device under test.
pos	Specifies the seek position.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.
x	Specifies the block number.

If an error occurs while writing to a temporary file during file system testing, the program displays:

```
write(fd, buf, size): message (Error n)
Unable to write block x
```

Variable	Description
fd	Specifies the device under test.
buf	Specifies the data buffer location.
size	Specifies the size of the block.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.
x	Specifies the block number.

If a read error occurs during this test, the program displays:

```
read(fd, buf, size): message (Error n)
Unable to read block x
```

Variable	Description
fd	Specifies the device under test.
buf	Specifies the data buffer location.
size	Specifies the size of the block.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual
n	Specifies the error number associated with a system error message.
x	Specifies the block number.

If the write checksum is not equal to the read checksum during random file system testing, the program displays:

```
Block: n, Expected Checksum: 0xe, Actual Checksum: 0xa
```

Variable	Description
n	Specifies the block number.
e	Specifies the expected checksum value.
a	Specifies the actual checksum value.

After testing is completed, the Disk Exerciser performs some cleanup operations. If the program is unable to unlink the temporary file, it displays:

```
unlink(file): message (Error n)
```

Variable	Description
file	Specifies the name of the temp file.
message	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the UMAX V Programmer's Reference manual
n	Specifies the error number associated with a system error message

If any errors were detected during this test, the program displays the following message with the other error messages:

```
Random File Test Failed
```

CHAPTER 6

Memory Exercisers

Two Memory Exercisers reside in `/usr/oe/test/memory`:

- Memory Exerciser (`memtest`)
- Read Only Memory Exerciser (`readmem`).

The Memory Exerciser verifies the integrity of a block of memory. Each test in the exerciser writes into this memory area using different data patterns and methods. After a test has written into memory, it reads the data and verifies that the data read equals the data written.

The Read Only Memory Exerciser allows reading any memory space at any time except the upper one megabyte of memory, which is reserved for the `readmem` process. The memory space used by the kernel may also be read.

Special Requirements

The Memory Exercisers have the following special requirements:

- If a user-defined Memory Type is specified for the Read Only Memory Exerciser, the starting address of the memory range must be page-bounded.
- The maximum work area size for the Address=Data Test in the Memory Exerciser must be word-bounded.
- The memory range tested by the Read Only Memory Exerciser can not exceed the value in the `maxumem` parameter defined in the kernel. If the buffer size is greater than the `maxumem` value, the program will adjust the size of the memory test range accordingly. The `maxumem` value can be dynamically changed with the `/etc/lbin/pagetune` utility or statically changed by modifying the `maxumem` parameter in the system file. As with any other tunable value, extreme caution should be used when modifying the default `maxumem` value because system performance may be affected.

Unique Parameters

In addition to the parameters described in Chapter 3, the Memory Exercisers provide the following unique test parameters.

TABLE 6-1 Memory Exerciser Parameters

Parameter	Description
Maximum Work Area Size	Enter the size of the memory work area to test. If the size entered cannot be allocated, the work area size will be determined by the value of the Work Area Unit Size parameter. If you enter 0, the exerciser attempts to allocate the largest block of memory possible, starting with the Work Area Unit Size and increasing in size by the Work Area Unit Size until allocation fails. The default maximum Work Area Size is 4,096 bytes (1 page).
Work Area Unit Size (Bytes)	Enter the size of the Work Area Unit in bytes. The size of the work area is increased or decreased by multiples of this value when attempting to allocate a work area. This value must be greater than zero. The default Work Area Unit Size is 1024 bytes (1 KB).
Fill Pattern	Enter the data pattern for the Pattern Fill Test. The default pattern is 0123456789ABCDEF.
Run Test Ganged	This parameter is used to enable or disable running tests ganged. If enabled, the test creates a gang and enters it when the pass starts, and then exits and deletes the gang after all enabled subtests have been run. Running ganged dedicates a processor to exclusively run the memory test.
Run Test Locked	This parameter is used to enable or disable running tests locked. If enabled, the test process attempts to lock itself down in memory, preventing itself from being paged in and out while it runs.

Note – If both the Gang and Lock options are enabled, any addresses reported during testing will be physical addresses. If not, all addresses reported will be virtual addresses.

If you want to increase the Maximum Work Area Size from its default value, set the Memory March Test to `SKIP`. This recommendation is based solely on execution time considerations.

TABLE 6-2 Read Only Memory Exerciser Parameters

Parameter	Description	
Memory Type	Select the type of memory to read. The following memory type selections are available:	
	Memory Type	**Function**
	Cpu	Read just CPU memory.
	Expansion	Read just Expansion memory.
	All Memory	Read all physical memory.
	User Defined	Read a user-defined memory range. The starting address of the memory range must be on a page boundary. Pages are 4096 bytes or 0x1000 bytes.
Starting Address (page bound)	Specify the starting address of the memory range to be read. This parameter is valid only if `User Defined` was selected as the Memory Type. The starting address must be on a page boundary.	
Buffer Size (bytes)	Specify the size of the memory range to be read. This parameter is valid only if `User Defined` was selected as the Memory Type.	

Note – The buffer size is limited to the value of the `maxumem` parameter defined in the kernel. If the user-defined buffer size exceeds this value, the program will adjust the size of the buffer accordingly.

The program reserves the upper one megabyte of memory for the `readmem` process.

Memory Exerciser Test Descriptions

The Memory Exerciser contains the following tests:

- Sequential Fill
- Interleave Fill
- Random Fill
- Shift Left
- Shift Right
- Fill Left
- Fill Right
- Complement
- Pattern Fill
- Address=Data
- Increment
- Decrement
- Memory March

Prior to test execution, the Memory Exerciser performs the following operations:

- Allocates a memory work area for testing.
- Creates and enters a gang if ganging is enabled. If successful, the program displays the following debug messages:

```
Gang created
Gang entered
Cpu id = id
```

Variable	Description
id	Specifies the CPU ID number.

- The program then locks the process into memory if locking is enabled, and displays the following debug message:

```
Process locked down in memory
```

In order to perform virtual to physical memory address translations, the program requires the CPU to be ganged and the process to be locked down in memory.

The program reports the starting address of the memory test area. If ganging and locking are enabled, the program displays the following debug and milestone messages:

- The program displays the following debug message:

```
Allocated x continuous bytes of virtual memory starting at 0xva
```

Variable	Description
x	Specifies the size of the memory work area in bytes.
va	Specifies the starting virtual address of the memory work area.

- The program then displays the following milestone message:

```
Allocated x continuous bytes of memory starting at physical address
0xpa
```

Variable	Description
x	Specifies the size of the memory work area in bytes.
pa	Specifies the physical starting address of the memory work area.

If ganging and locking are disabled, the program displays the following milestone message:

```
Allocated x continuous bytes of memory starting at virtual address
0xva
```

Variable	Description
x	Specifies the size of the memory work area in bytes.
va	Specifies the starting virtual address of the memory work area.

After all the tests execute, the Memory Exerciser performs the following test cleanup operations:

- Reports the total number of errors encountered during testing with the following milestone message:

```
Encountered a total of x errors
```

Variable	Description
x	Specifies the total number of errors encountered during Memory Exerciser testing.

- Frees up the allocated memory work area space.
- Exits from and destroys the gang if a gang was created.
- Unlocks the process from memory if locking was enabled.

The following sections describe the operations performed by each of the Memory Exerciser tests.

Sequential Fill Test

This test fills the memory work area from the lowest address to the highest address with the following data patterns: 0xFF, 0, 0xFF, 0xA5, 0x5A, 0xA5, 0xC3, 0x3C, 0xC3, 0xE1, 0x1E, 0xE1, 0xD2, 0x2D, 0xD2, 0xB4, 0x4B, 0xB4, 0x96, 0x69, 0x96, 0x87, 0x78, 0x87.

The test performs the following operations:

- Zeros each location in the memory work area.
- Displays the following milestone message to identify the data byte pattern to be used for the fill:

```
Filling memory with value 0xpp
```

Variable	Description
pp	Specifies the test data pattern.

- Fills the work area with a data pattern. As each location is written, the following debug message is displayed:

```
Writing 0xpp to {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
pp	Specifies the byte data pattern.
aaaaaaaa	Specifies the memory location being written.

- Reads the data from memory and verifies that the data read equals the data written.
- Changes to the next data pattern and repeats the fill, read, and verify sequence.

The test continues filling and checking memory until all data patterns have been used. If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Sequential Fill Test
```

Variable	Description
x	specifies the number of errors detected.

Interleave Fill Test

This test accesses the memory work area from the lowest address to the highest address with increasing interleave values. The test starts with an interleave value of 2, and increments by one until a maximum interleave value of 7 is reached. Each time the test fills the work area, the data is read and verified before the next interleave value is used.

The test performs the following operations:

- Zeros each location in the memory work area.
- Identifies the interleaving value under test with the following milestone message:

```
Testing interleave of x
```

Variable	Description
x	Specifies the interleave value under test.

- Writes the data to memory. As each location is written, the following debug message is displayed:

```
Writing 0xpp to {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
pp	Specifies the byte data pattern.
aaaaaaaa	Specifies the memory location being written.

- Reads the data from memory and verifies the data read equals the data written.
- Changes to the next interleaving value and repeats the testing process.

The test continues writing and checking memory until all interleave values have been used. If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Interleave Fill Test
```

Variable	Description
x	Specifies the number of errors detected.

The test iterates a maximum of six times over the work area. If the work area specified is less than eight bytes, the maximum interleave value will equal one less than the maximum work area size.

Random Fill Test

This test fills the memory work area in random order with a data pattern of 0xA5. After the work area is filled, the test reads the data from memory and verifies that the data read equals the data written.

The test performs the following operations:

- Zeros each location in the memory work area.
- Writes the data to memory and identifies the memory location being written with the following debug message:

```
Writing to {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location being written.

- Reads the data from memory and verifies the data read equals the data written.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Random Fill Test
```

Variable	Description
x	Specifies the number of errors detected.

Shift Left Test

This test performs the following operations:

- Zeros each location in the work area.
- Sets the low-order bit in each location. The following milestone message identifies the current data pattern:

```
Shifting to value 0xpp
```

Variable	Description
pp	Specifies the current byte data pattern.

- Shifts the set bit in each location one position to the left. The following debug message is displayed to identify the memory location under test:

```
Shifting {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies that only the desired bit is set.
- Repeats the left shift until each bit in each location has been tested.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Shift Left Test
```

Variable	Description
x	Specifies the number of errors detected.

Shift Right Test

This test performs the following operations:

- Zeros each location in the work area.
- Sets the high-order bit in each location. The following milestone message identifies the current data pattern:

```
Shifting to value 0xpp
```

Variable	Description
pp	Specifies the current byte data pattern.

- Shifts the set bit in each location one position to the right. The following debug message is displayed to identify the memory location under test:

```
Shifting {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies that only the desired bit is set.
- Repeats the right shift until each bit in each location has been tested.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Shift Right Test
```

Variable	Description
x	Specifies the number of errors detected.

Fill Left Test

This test performs the following operations:

- Zeros each location in the work area.
- Sets the low-order bit in each location. The following milestone message identifies the current data pattern:

```
Shifting to value 0xpp
```

Variable	Description
pp	Specifies the current byte data pattern.

- Shifts the set bit in each location one position to the left. The bit set previously is kept set. The following debug message is displayed to identify the memory location under test:

```
Shifting {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies that only the desired bits are set.
- Repeats the left shift until all the bits in each location are set.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Fill Left Test
```

Variable	Description
x	Specifies the number of errors detected.

Fill Right Test

This test performs the following operations:

- Zeros each location in the work area.
- Sets the high-order bit in each location. The following milestone message identifies the current data pattern:

```
Shifting to value 0xpp
```

Variable	Description
pp	Specifies the current byte data pattern.

- Shifts the set bit in each location one position to the right. The bit set previously is kept set. The following debug message is displayed to identify the memory location under test:

```
Shifting {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies that only the desired bits are set.
- Repeats the right shift until all the bits in each location are set.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Fill Right Test
```

Variable	Description
x	Specifies the number of errors detected.

Complement Test

This test performs the following operations:

- Zeros each location in the work area.
- Fills each location in the memory work area with a data pattern. The following milestone message identifies the original data pattern:

```
Testing complement of 0xpp
```

Variable	Description
pp	Specifies the original byte data pattern.

- Logically negates the contents of each location. As each location is negated, the following debug message is reported:

```
Complementing {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies each location contains the correct data.
- Repeats the fill, negate and verify sequence until all data patterns have been used.

This test uses the following data patterns: 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Complement Test
```

Variable	Description
x	Specifies the number of errors detected.

Pattern Fill Test

This test zeros the memory work area and then fills it with a user-supplied data pattern. The data pattern is specified when the Memory Exerciser test parameters are selected. After filling memory, the test reads and verifies the data.

If the data pattern is smaller than the work area, the pattern is repeated throughout the work area.

Before the test starts, the following milestone message identifies the data pattern used during the test:

```
Filling memory with repeated 'pat'
```

Variable	Description
pat	Specifies the data fill pattern.

As each memory location is written, the following debug message is displayed:

```
Filling {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Pattern Fill Test
```

Variable	Description
x	Specifies the number of errors detected.

Address=Data Test

This test zeros the memory work area and then writes the address of each location in the work area as data into the same location. After all locations are filled with their respective addresses, the test reads and verifies the contents of each location.

As each memory location is filled, the test displays the following debug message:

```
Filling {physical|virtual} address 0xaaaaaaaa with 0xpppppppp
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.
pppppppp	Specifies the data pattern being written into the memory location.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Address=Data Test
```

Variable	Description
x	Specifies the number of errors detected.

Note – This test requires the maximum work area size to be word-bounded.

Increment Test

This test performs the following operations:

- Zeros each location in the work area.
- Increments the contents of each location by one to reach the following data values: 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10. The test then increments by 0x10 to reach the following data values: 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0xA0, 0xB0, 0xC0, 0xD0, 0xE0, 0xF0. Before incrementing the memory locations, the test displays the following milestone message:

```
Incrementing to 0xpp
```

Variable	Description
pp	Specifies the byte data value being written into memory.

As each location is incremented, the test displays the following debug message:

```
Incrementing {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies each location was incremented.
- Repeats the increment and verify sequence until each location has been incremented to reach the end of the data values listed above.

Unlike the Sequential Fill Test, which changes the contents of the work area by writing data patterns taken from outside the area, this test increments each location directly.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Increment Test
```

Variable	Description
x	Specifies the number of errors detected.

Decrement Test

This test performs the following operations:

- Zeros each location in the work area.
- Decrements the contents of each location by one to reach the following data values: 0xFF, 0xFE, 0xFD, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8, 0xF7, 0xF6, 0xF5, 0xF4, 0xF3, 0xF2, 0xF1, 0xF0. The test then decrements by 0x10 to reach the following data values: 0xE0, 0xD0, 0xC0, 0xB0, 0xA0, 0x90, 0x80, 0x70, 0x60, 0x50, 0x40, 0x30, 0x20, 0x10, 0. Before decrementing the memory locations, the test displays the following milestone message:

```
Decrementing to 0xpp
```

Variable	Description
pp	Specifies the byte data value being written into memory.

As each location is decremented, the test displays the following debug message:

```
Decrementing {physical|virtual} address 0xaaaaaaaa
```

Variable	Description
aaaaaaaa	Specifies the memory location under test.

- Verifies each location was decremented.
- Repeats the decrement and verify sequence until each location has been decremented to reach the end of the data values listed above.

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Decrement Test
```

Variable	Description
x	Specifies the number of errors detected.

Memory March Test

This test performs the following operations:

- Zeros each location in the work area and displays the following milestone message:

```
Begin March
```

- Copies the value from one location to the next, and so on, until the original data value in the starting location has been marched through the entire work area. The test displays the following milestone message to identify its location in the march iterations:

```
March Pass n
```

Variable	Description
n	Specifies the number of this iteration. The total number of iterations equals one less than the maximum work area size.

- As each value is marched through memory, the test displays a debug message in the following format:

```
0xpp moved to {virtual|physical} address 0xaaaaaaaa
```

Variable	Description
pp	Specifies the data value marched through the work area.
aaaaaaaa	Specifies the address receiving the data.

- After all iterations are complete, the test displays a debug message in the following format:

```
Last Loc Data = 0xll  First Loc Data = 0xff
```

Variable	Description
ll	Specifies the data value in the last memory location tested.
ff	Specifies the data value in the first memory location tested.

The initial value loaded into the first location should end up in the last location.

- The test then displays the following milestone message and verifies the data in each location:

```
Begin Verify
```

If any errors were detected during the test, the following milestone message is displayed:

```
Encountered x errors during the Memory March Test
```

Variable	Description
x	Specifies the number of errors detected.

Read Only Memory Exerciser Test Descriptions

The Read Only Memory Exerciser contains a single Read Memory Test. Once started, the Exerciser performs the following routines:

- Initialization
- Test Execution
- Cleanup

Initialization

The Read Only Memory Exerciser performs the following operations during program initialization:

- Starts checking the machine configuration and displays the following debug message:

```
In config routine
```

- Determines the system memory size and configuration. The program displays the following debug messages identifying the memory configuration and size:

```
Slot 0: CPU Configuration: config
Slot n: Expansion Memory size
Total Size of Expansion Memory = x MB
Total Size of User Memory = pgcnt Pages
Total Size of User Memory = xxx MB
```

Variable	Description
config	Specifies the CPU configuration.
n	Specifies the number of the slot containing the Expansion Memory board. This message is repeated for each Expansion Memory board.
size	Specifies the size (in megabytes) of the Expansion Memory on this board.
x	Specifies the total size (in megabytes) of Expansion Memory in the system.
pgcnt	Specifies the maximum virtual memory allowed per process. This is the value of the `maxumem` parameter in pages.
xxx	Specifies the maximum virtual memory allowed per process expressed in megabytes.

- Displays the following debug message:

```
Total Physical Memory Size: xxx MB
```

Variable	Description
xxx	Specifies the total size (in megabytes) of physical memory in the system.

- Checks the user-selected Memory Type.
  - If `User Defined` is specified, the program verifies that the requested memory range is less than or equal to the total system memory size. The program then verifies that the starting address is on a page boundary. The program determines the lower boundary of system memory, verifies that the start address is equal to or greater than the lower boundary, and displays the following debug messages:

```
lower boundary = 0xllllllll
Start Address = 0xstartaddr
```

Variable	Description
llllllll	Specifies the lower boundary of system memory.
startaddr	Specifies the first address in the user-selected memory test range.

The program then determines the upper boundary of system memory, verifies that the last address is less than or equal to the upper boundary, and displays the following debug messages:

```
upper boundary = 0xuuuuuuuu
Last Address = 0xlastaddr
```

Variable	Description
uuuuuuuu	Specifies the upper boundary of system memory.
lastaddr	Specifies the last address in the user-selected memory test range.

The program will exit if either boundary test fails.

  - If `Cpu` is specified, the program determines the size of CPU memory, sets the starting address and buffer size, and displays the following debug messages:

```
Start Address = 0xstartaddr
Last Address = 0xlastaddr
```

Variable	Description
startaddr	Specifies the first address in CPU memory.
lastaddr	Specifies the last address in CPU memory.

- If `Expansion` is specified, the program sets the starting address and buffer size for expansion memory testing and displays the following debug messages:

```
Start Address = 0xstartaddr
Last Address = 0xlastaddr
```

Variable	Description
startaddr	Specifies the first address in Expansion Memory.
lastaddr	Specifies the last address in Expansion Memory.

- If `All Memory` is specified, the program sets the starting address and buffer size for all of physical memory and displays the following debug messages:

```
Start Address = 0xstartaddr
Last Address = 0xlastaddr
```

Variable	Description
startaddr	Specifies the first address in system memory.
lastaddr	Specifies the last address in system memory.

Note – If the buffer size is greater than the value in the `maxumem` parameter, the program adjusts the size of the memory test range and displays the following debug messages:

```
Attempted to allocate xxxxxxxx
Actually Allocated yyyyyyyy
```

Variable	Description
xxxxxxxx	Specifies the requested buffer size.
yyyyyyyy	Specifies the buffer size allocated by the program.

The program then displays the following error message and continues execution:

```
Unable to Allocate Amount of Memory Requested
```

- Creates and enters a gang.
- Gets the ID number for the CPU which has been ganged and displays the following debug message:

```
Cpu Id : n
```

Variable	Description
n	Specifies the ID number for the ganged CPU.

- Checks the device status.
- Performs a `phys_permit()` system call.
- Performs a `phys_map()` system call and displays the following debug message:

```
Memory token = xxx
```

Variable	Description
xxx	Specifies the token returned by the `phys_map()` system call.

- Displays one of the following milestone messages identifying the type of memory to be tested:

```
Testing Expansion Memory
Testing Cpu Memory
Testing Cpu and Expansion Memory
```

- Displays the following debug message showing the virtual address returned by the `phys_map()` system call:

```
virtual attach_address = 0xaddress
```

Variable	Description
address	Specifies the virtual address returned by the system call.

- Displays the following milestone messages showing the address range to be tested:

```
Starting Physical Address = 0xstartaddr
Ending Physical Address = 0xlastaddr
```

Variable	Description
startaddr	Specifies the first address in the memory test range.
lastaddr	Specifies the last address in the memory test range.

Test Execution

During test execution, the Read Only Memory Exerciser performs the following operations:

- Initializes a pointer to the starting memory address and displays the following milestone message:

```
Read Memory Test Start
```

- Arms the timeout alarm.
- Reads bytes from all locations in the memory test range. If the `User Defined` memory type is selected and debug messages are enabled, the program displays the following debug message as each location is read:

```
Physical Address = 0xaddr   Data = 0xbb
```

Variable	Description
addr	Specifies the physical address being read.
bb	Specifies the data read from this location.

- Checks for a timeout condition. If the read completed successfully, the program displays the following milestone message:

```
Read of memory completed successfully.
```

If a timeout occurred, the program posts an error message.

Cleanup

After test execution is completed, the Read Only Memory Exerciser performs the following test cleanup operations:

- Performs a `gang_exit()` system call.
- Performs a `gang_destroy()` system call.
- Performs a `phys_detach()` system call.

Memory Exerciser Error Messages

If an error occurs during Memory Exerciser operation, one of the following types of errors is reported:

- Initialization
- Memory
- Interleave
- `gang()`
- `plockres()`

Initialization Error Messages

If the Memory Exerciser is unable to allocate a work area for testing, the program displays:

```
Couldn’t allocate a work area in memory
```

If the Work Area Unit Size parameter is zero, the program displays:

```
No work area in memory was allocated
```

If no fill pattern was specified for the Pattern Fill Test, the program displays:

```
No pattern to fill memory with
```

If the maximum memory work area is not word-bounded for the Address=Data Test, the program displays:

```
Maximum work area size must be word bounded.
```

Memory Error Messages

If a memory error is encountered during the execution of the Memory Exerciser, the error is reported with one of the following messages:

```
{Physical|Virtual} Address = 0xbb, Expected = 0xee, Actual = 0xaa
{Physical|Virtual} Address = 0xbb, Expected = 0xee, Encountered =
0xaa
```

Variable	Description
bb	Specifies the failing memory location.
ee	Specifies the expected data value.
aa	Specifies the actual data value received.

Interleave Error Messages

If a memory error is encountered during the Interleave Fill Test, the error is reported with the following message:

```
{Physical|Virtual} Address = 0xbb,Expected = 0xee,Actual =
0xaa,Interleave = 0xii
```

Variable	Description
bb	Specifies the failing memory location.
ee	Specifies the expected data value.
aa	Specifies the actual data value received.
ii	Specifies the interleaving factor.

`gang()` Error Messages

The following error messages are associated with running ganged:

- If running ganged was enabled, but the `gang_create(2)` system call failed:

```
Unable to create gang. Errno = n
```

- If running ganged was enabled and the gang was successfully created, but the `gang_enter(2)` system call failed:

```
Unable to enter gang. Errno = n
```

- If the test completed all subtests, but the `gang_exit(2)` system call failed:

```
Unable to exit from the gang. Errno = n
```

- If the test was able to exit the gang, but the `gang_destroy(2)` system call failed:

```
Unable to destroy gang. Errno = n
```

Variable	Description
n	Specifies the error number associated with the failing system call. System call errors are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.

`plockres()` Error Messages

The following error messages are associated with running locked:

- If running locked was enabled, but the `plockres()` system call failed while attempting to lock down the process into memory:

```
Unable to lock process into memory. Errno = n
```

- If the subtests ran in locked-down memory, but the `plockres()` system call failed while attempting to unlock the process from memory:

```
Unable to unlock process from memory. Errno = n
```

Read Only Memory Exerciser Error Messages

If an error occurs during Read Only Memory Exerciser operation, one of the following types of errors is reported:

- Initialization
- Read Memory
- System Call

Initialization Error Messages

If an error is detected during the initialization routine, the Read Only Memory Exerciser reports one of the following error messages and exits:

- If the program is unable to determine the machine type:

```
Init routine unable to determine machine type
```

- The following error messages are associated with the `User Defined` memory type:
  - If the buffer size specified by the user is larger than the amount of physical memory in the system:

```
Buffer Size exceeds the size of physical memory
```

  - If the user-specified start address is not on a page boundary:

```
The starting address must be page bounded
```

- If the program is unable to determine the lower boundary of memory:

```
Unable to determine the lower boundary of memory
```

- If the user specifies an invalid starting address:

```
Starting Address is outside the lower boundary of memory
```

- If the specified memory range is greater than the upper memory address boundary:

```
Memory Range exceeds upper address boundary
```

- If the memory type is undefined:

```
Don't understand memory type to be tested
```

- If the selected memory type is `Cpu` or `All Memory` and the program is unable to determine the size of CPU memory:

```
Unable to determine the cpu memory test range
```

- If the buffer size exceeds the value of the `maxumem` parameter, the program adjusts the size of the memory test range and displays:

```
Unable to Allocate Amount of Memory Requested
```

Read Memory Error Messages

If a timeout occurs during the execution of the Read Memory Test, the program displays:

```
Timeout occurred. Physical Address = 0xaddr
```

Variable	Description
addr	Specifies the last physical address accessed.

The timeout is set for a true CPU processing time of 5 minutes.

System Call Error Messages

If an error occurs during the execution of a system call, one of the following error messages is reported:

- If the program was unable to create a gang:

```
Unable to gang_create. Errno = n
```

- If the program was unable to enter the gang:

```
Unable to gang_enter. Errno = n
```

- If the program was unable to exit the gang:

```
Unable to exit from gang. Errno = n
```

- If the program was able to exit the gang, but the `gang_destroy()` system call failed:

```
Unable to destroy gang. Errno = n
```

- If a device status failure occurs during program initialization:

```
stat on /dev/vsi failed. Errno = n
```

or

```
stat on /dev/nbus failed. Errno = n
```

- If the `phys_permit()` system call fails:

```
Unable to phys_permit. Errno = n
```

- If the `phys_map()` system call fails:

```
Unable to phys_map. Errno = n
```

- If the `phys_detach()` system call fails:

```
Unable to phys_detach memory region. Errno = n
```

Variable	Description
n	Specifies the error number associated with the failing system call. System call errors are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.

CHAPTER 7

Network Exerciser

The Network Exerciser verifies system to system communications across a standard network connection. This exerciser uses the remote shell (`nsh`) command to test the network.

Special Requirements

This exerciser has the following special requirements:

- A standard UMAX V TCP/IP network.
- A remote system accessible with the `nsh()` command. The name of the remote system must be listed in `/etc/hosts`, `/.rhosts` and `/etc/hosts.equiv` on the local system. The remote system must be a "trusted host."
- Sufficient space must be available in the `/tmp file` system for work files.

Unique Parameters

In addition to the parameters described in Chapter 3, the Network Exerciser provides the following unique test parameter.

TABLE 7-1 Network Exerciser Parameters

Parameter	Description
Remote Host	Enter the host name of the system with which you want to communicate. The default host name is the name of the system running the Online Exerciser (the local host).

Test Descriptions

The Network Exerciser contains one test, which is a script file of UMAX V commands. This test script uses the `nsh` command to copy files from one system to another. The test performs the following operations:

- Logs onto the remote system.
- Transmits a file to and from the remote system.
- Compares the received file with the transmitted file.

Prior to starting the test script execution, the Network Exerciser displays the following debug message:

```
host hostname
```

Once the test script begins executing, it displays the following milestone message:

```
host - hostname
```

Variable	Description
hostname	Specifies the name of the remote host.

Error Messages

If an error is detected, one of the following types of error messages is reported:

- Remote Host
- File Transmit
- File Comparison

Remote Host Error Messages

If an invalid hostname was specified and the `ping` command failed, the program displays:

```
Unable to ping host 'hostname'.
```

If no response is received from the remote system, the program displays:

```
Host 'hostname' is not responding.
```

If the program is unable to log onto the remote system, the program displays:

```
Unable to remotely log onto host 'hostname'.
```

Variable	Description
hostname	Specifies the name of the remote system.

File Transmit Error Messages

If the program is unable to transmit the file to the remote system, it displays:

```
Unable to transmit file.
```

This error message indicates the failure of the `nsh` *hostname* `cat <`*in*`>` *out* command.

Variable	Description
hostname	Specifies the name of the remote system.
in	Specifies the name of the original file.
out	Specifies the name of the file received from the remote system.

File Comparison Error Messages

If the received file uncompressed successfully, but does not match the original file, the program displays:

```
Received file does not match transmitted file.
```

CHAPTER 8

Tape Exerciser

The Tape Exerciser (`tapetest`) resides in `/usr/oe/test/tape`. This exerciser attempts to exercise the selected tape drive by performing the following operations:

- Read and write data.
- Rewind tape.
- Copy data to and from tape using UMAX V commands.

Special Requirements

The Tape Exerciser has the following special requirements:

- A tape drive.
- A scratch tape is required for all testing. The Tape Exerciser destroys any data on the tape in the selected tape drive.
- Sufficient space must be available in the `/tmp` file system for work files.

Unique Parameters

In addition to the parameters described in Chapter 3, the Tape Exerciser provides the following unique test parameters.

TABLE 8-1 Tape Exerciser Parameters

Parameter	Description
Execute Sequential Writes and Reads	Use this parameter to select the Sequential Writes and Reads Test for execution. Valid selections are `Skip` and `Execute`. The default selection is `Execute`.
Execute Cpio / Dd / Tar Test	Use this parameter to select the CPIO / DD / TAR Test for execution.Valid selections are `Skip` and `Execute`. The default selection is `Skip`.

Logical Tape Drive	Enter the path to the drive under test in one of the following formats:
	
	`/dev/rmt/#h`
	`/dev/rmt/v#h`
	
	# specifies a tape drive number. The default tape drive is `/dev/rmt/0h.` The second format is for tape drives connected to VME controllers.

Test Descriptions

The Tape Exerciser is divided into the following tests:

- Tape Sequential Writes and Reads Test
- CPIO / DD / TAR Test

Tape Sequential Writes and Read Test

This test performs the following operations:

- Opens the selected tape.
- Rewinds the tape.
- Writes 200 blocks of data to the tape. Each block is 512 bytes. Before performing the write operation, the test displays the following debug message:

```
write byte count - n
```

Variable	Description
n	Specifies the number of bytes to be written.

- Rewinds the tape.
- Reads the 200 blocks from the tape and verifies that the data read equals the data written. Before performing the read operation, the test displays the following debug message:

```
read byte count - n
```

Variable	Description
n	Specifies the number of bytes to be read.

- Rewinds the tape.
- Closes the selected tape.

- Waits for the time specified in the `Delay` parameter and displays the following milestone message:

```
Going to Sleep for xxxx Seconds
```

Variable	Description
xxxx	Specifies the delay time in seconds.

CPIO / DD / TAR Test

This test is a tape script program used to verify the ability of the tape drive to function correctly under control of the UMAX V operating system. The test contains the following subtests:

- CPIO Test
- DD Test
- TAR Test

Prior to starting the test script, the Tape Exerciser displays the following debug message:

```
unit n
```

Variable	Description
n	Specifies the tape unit under test.

During tape script execution, the program displays the following milestones:

```
unit - n
b blocks
x records in
y records out
```

Variable	Description
n	Specifies the tape unit under test.
b	Specifies the number of blocks transferred during the CPIO test.
x	Specifies the number of records input during the DD test.
y	Specifies the number of records output during the DD test.

CPIO Test

This test performs the following operations:

- Displays the following milestone message:

```
CPIO Test Start
```

- Copies the `/etc` directory to the tape using the `cpio` (high density) command. The tape is not rewound after this operation.
- Copies the `/etc` directory to the tape using the `cpio` (high density) command and rewinds the tape.
- Copies the file `group` from the tape to a temporary file with the `cpio` command.
- Compares the file copied from the tape to the original file (`/etc/group`) using the `diff` command.
- Deletes the newly created file.
- Displays the following milestone message:

```
CPIO Test Complete
```

DD Test

This test performs the following operations:

- Displays the following milestone message:

```
DD Test Start
```

- Copies the file `/unix` to tape using the `dd` command. The tape is not rewound after this operation.
- Repeats the previous step once and rewinds the tape.
- Copies the file `/unix` from the tape using the `dd` command.
- Uses the `cmp` command to compare the original file with the copy.
- Deletes the newly created file.
- Displays the following milestone message:

```
DD Test Complete
```

TAR Test

This test performs the following operations:

- Displays the following milestone messages:

```
TAR Test Start
TAR: blocksize = n
```

Variable	Description
n	Specifies the blocksize of the tar image created.

- Copies the `/etc` directory to the tape using the `tar` command. The tape is not rewound after this operation.
- Repeats the previous step once and rewinds the tape.
- Extracts the `/etc/group` file from the tape using the `tar` command.
- Compares the new group file with the original using the `diff` command.
- Removes all newly created files and directories.
- Displays the following milestone message:

```
TAR Test Complete
```

Error Messages

If an error occurs during Tape Exerciser operation, an error message is sent to the log file. The following types of errors are possible:

- Initialization
- Rewind
- Write
- Read
- Data Comparison
- File List Generation

Initialization Error Messages

If an error is detected during the initialization of the device under test, the Tape Exerciser reports the error with one of the following messages:

- If the Tape Exerciser is unable to obtain the tape configuration:

```
Unable to obtain tape configuration.
```

- If the Tape Exerciser is unable to open the unit under test:

```
Unable to open device drive -- Error n
```

Variable	Description
drive	Specifies the tape drive under test.
n	Specifies the error number associated with the failing system call. System call errors are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.

Rewind Error Messages

If a rewind error occurs, the program displays:

```
Rewind Error on Tape drive -- Error  n
```

Variable	Description
drive	Specifies the tape drive under test.
n	Specifies the error number associated with the failing system call. System call errors are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.

Write Error Messages

If an error occurs while writing to the tape during the Sequential Writes and Reads Test, the following message is reported:

```
Write Error Code cccc
```

Variable	Description
cccc	Specifies an error code representing the type of system error detected.

If an error occurs while writing to the tape when running the CPIO / DD / TAR Test, one of the following errors is reported:

- If the error occurred while executing a `cpio` command, one of the following error messages is reported. The first message indicates the failure occurred when writing the first image to tape. The second message is associated with writing the second image:

```
CPIO: Error writing to drive drivehn
CPIO: Error writing to drive driveh
```

- If the error occurred while executing a `dd` command, one of the following error messages is reported. The first message indicates the failure occurred when writing the first image to tape. The second message is associated with writing the second image:

```
DD: Error writing to drive drivehn
DD: Error writing to drive driveh
```

- If the error occurred while executing a `tar` command, one of the following error messages is reported. The first message indicates the failure occurred when writing the first image to tape. The second message is associated with writing the second image:

```
TAR: Error writing to drive drivehn
TAR: Error writing to drive driveh
```

Variable	Description
drive	Specifies the tape drive under test when the error occurred.

Read Error Messages

If an error occurs while reading from tape during the Sequential Writes and Reads Test, the program displays:

```
Read Error Code cccc
```

Variable	Description
cccc	Specifies an error code representing the type of system error detected.

If an error occurs while reading from the tape when running the CPIO / DD / TAR Test the program displays one of the following messages:

- If the error occurred while reading the first image written with a `cpio` command:

```
CPIO: Error reading from drive driveh
```

- If the error occurred while reading the first image written with a `dd` command:

```
DD: Error reading from drive drive#h.
```

- If the error occurred while reading the first image written with a `tar` command:

```
TAR: Error reading from drive drive#h.
```

Variable	Description
drive	Specifies the tape drive under test.

Data Comparison Error Messages

If a data error is detected while reading and verifying data written to the tape during the Sequential Writes and Reads Test, the program displays:

```
Read Data Error expected edata, actual adata
```

Variable	Description
edata	Specifies the expected data value.
adata	Specifies the actual data value.

If the data read from tape is not equal to the data written to tape during the execution of the CPIO / DD / TAR Test, the program displays one of the following messages:

- If the error is detected while executing a `diff` command on a file written with the `cpio` command:

```
CPIO: Read file does not match original file.
```

- If the error is detected while executing a `cmp` command on a file written with the `dd` command:

```
DD: Read file does not match original file.
```

- If the error is detected while executing a `diff` command on a file written with the `tar` command:

```
TAR: Read file does not match original file.
```

File List Generation Error Messages

If the CPIO Test is unable to generate a list of files to write with the `cpio` command, the program displays:

```
CPIO: Error during file list creation
```

This error indicates a failure of the `find(1)` command.

CHAPTER 9

UMAX V Exerciser

The UMAX V Exerciser tests the ability of the UMAX V operating system to execute commands and system calls and measures system performance by timing various operations.

Special Requirements

There are no special requirements for running the UMAX V Exerciser.

Test Descriptions

The UMAX V Exerciser is divided into the following individual routines:

- `tpid`
- `tindex`
- `tproc`
- `tyld`
- `yieldtest`
- `plock_test`
- `plock_test2`
- `cctl_test`
- `cctl_test2`
- `cctl_test3`
- `async_test`
- `shmtest`
- `block_test2`

tpid

The tpid routine establishes the minimum times for executing system calls. This task determines how long it takes to execute the `getpid()` system call and reports the acquired times in the following format:

```
n getpid()s in x usecs
y usecs / getpid()
```

Variable	Description
n	Specifies the number of `getpid()` calls executed.
x	Specifies the time required to execute *n* `getpid()` calls.
y	Specifies the time required to execute one `getpid()` call.

tindex

The tindex routine establishes the times for executing the `get_index()` system call. This task determines how long it takes to execute the system call and reports the acquired times in the following format:

```
Total ms l overhead o
Time for n get_index() x micro-seconds
y usecs / get_index()
```

Variable	Description
l	Specifies the time required to execute the timing loop.
o	Specifies the overhead time involved in executing the loop.
n	Specifies the number of `get_index()` calls executed.
x	Specifies the time required to execute *n* `get_index()` calls.
y	Specifies the time required to execute one `get_index()` call.

tproc

The `tproc` routine derives timing values and exercises the `res_ctl()` system call. This task uses the `fork()` system call to create 1000 processes with 5MB of memory. All the processes share their virtual space. After exercising large portions of the kernal code, the program calculates the time required for each `fork()`. Error reports may be generated by problems in either the child process or the timing loop. This task reports the following milestones:

```
1000 iterations of fork
5Mb private data
Nunix(07 = [-, -, -, -, -, Text, Data, Stack])
```

The task reports the execution time in the following format:

```
s seconds/fork
```

Variable	Description
s	Specifies the time calculated for the execution of each `fork()` call.

yieldtest

The `yieldtest` routine verifies the functionality of the `yield()` system call. This task uses `fork()` to create multiple child processes. All processes, including the parent process, execute the `yield()` call. This task reports the following milestones:

```
Yield test 10000 repetitions ...
Yieldtest: Success
```

tyld

The tyld routine determines the execution time for `yield()` system calls. The task first creates a gang to increase execution speed. The task reports the execution milestones in the following format:

```
In Gang g
n yield()s in  x usecs
y usecs / yield()
```

Variable	Description
g	Specifies the gang identification number.
n	Specifies the number of `yield()` calls executed.
x	Specifies the time required to execute *n* `yield()` calls.
y	Specifies the time required to execute one `yield()` call.

plock_test

The `plock_test` routine verifies the functionality of the `plock()` system call. The task starts running without superuser privileges to verify that the `plock()` call cannot be executed. The task then switches to superuser and uses three test sequences to verify that the `plock()` call functions correctly. The task reports the following milestone:

```
Testing plock running privileged
```

The task then begins the first test sequence.

Test Sequence 1

The task executes three `plock()` system calls with the following variables:

- `TXTLOCK`
- `PROCLOCK`
- `UNLOCK`

The first call should function correctly but the second call should fail because a test lock already exists on the calling process. The third call should also function correctly. This sequence reports the following milestone:

```
test sequence #1 ...
```

Test Sequence 2

The task executes three `plock()` system calls with the following variables:

- `DATLOCK`
- `PROCLOCK`
- `UNLOCK`

The first call should function correctly but the second call should fail because a data lock already exists on the calling process. The third call should also function correctly. This sequence reports the following milestone:

```
test sequence #2 ...
```

Test Sequence 3

The task executes five `plock()` system calls with the following variables:

- `PROCLOCK`
- `DATLOCK`
- `TXTLOCK`
- `UNLOCK`
- `UNLOCK`

The first call should execute correctly but the data and text lock calls should fail because a process lock already exits. The first unlock should execute correctly but the second unlock should fail because no type of lock exists. This sequence reports the following milestone:

```
test sequence #3 ...
```

plock_test2

The `plock_test2` routine verifies the execution of the `plockres()` system call by timing memory accesses. The task first accesses a small number of untouched pages and calculates the access time. The task then uses the `plockres()` call to force a second set of untouched pages to become memory resident and verifies that the memory access time is faster for the second set of pages. This task reports the following milestones:

```
Untouched memory: u
After plockres  : a
```

Variable	Description
u	Specifies the access time for the untouched pages.
a	Specifies the access time for the memory resident pages.

cctl_test

The `cctl_test` routine verifies the ability of the `cachectl()` system call to enable and disable caching and verifies that accesses with caching enabled are faster than accesses with caching disabled. The task accesses a small array using the `cachectl()` call with the `CCTL_ON` and `CCTL_OFF` commands and compares the access times. This task reports the following milestones:

```
4096 iterations with memory locked
Cache Enabled: x seconds
Cache Disabled: y seconds
Cache Enabled: x seconds
Cache Disabled: y seconds
```

Variable	Description
x	Specifies the times for memory accesses with cache enabled.
y	Specifies the times for memory accesses with cache disabled.

The first message identifies the number of times the memory access test is performed to calculate the memory access times.

cctl_test2

The `cctl_test2` routine verifies the ability of the `cachectl()` system call to protect a memory segment by setting it to read only access. The task first sets up a shared data segment and executes the `cachectl()` call with the `CCTL_RONLY` command. The task then uses the `fork()` system call to create a child process. The child attempts to perform the following operations:

- Read and validate the data segment
- Write to the data segment

Successful completion of the write operation indicates a failure because the memory should be set to read only access.

cctl_test3

The `cctl_test3` routine verifies the ability of the `cachectl()` system call to make an address range memory resident. The task first times the memory access for a set of untouched pages. The task then executes the `cachectl()` system call with the `CCTL_PLOCK` command to force a second set of untouched pages to become memory resident. Finally, the task determines the access time for the second set of pages and verifies that access is faster when the pages are memory resident. This test reports the following milestones:

```
Untouched memory: x seconds
After cachectl  : y seconds
```

Variable	Description
x	Specifies the access time for the untouched pages.
y	Specifies the access time for the memory resident pages.

async_test

The `async_test` routine calculates the execution times for using the `block()` and `unblock()` system calls to start a child process used to perform I/O for a ganged, locked process. The parent process performs the following operations:

- Reports the following milestone:

```
async_test.id: Rd+wflag  kflag  -n reps  (size bbbbKb) -f filename
```

Variable	Description
id	Specifies the process identification number.
wflag	Specifies the kid write flag (`Wr`).
kflag	Specifies the kid gang flag (`Kidgang`).
reps	Specifies the number of repetitions.
bbbb	Specifies the size of the disk transactions.
filename	Specifies the name of the file used for I/O.

- Sets up a timing module for timing system calls.
- Executes the `plockres(PROCLOCK)` system call to lock memory.
- Uses two processors to execute the `gang_create` system call to create a gang of one process.
- Executes the `gang_enter` system call to enter the gang.
- Uses the `shmget()` and `shmat()` system calls to set up the shared memory space for both the parent and child processes.
- Uses the `fork()` system call to create a child process.

The child process then performs the following operations:

- Executes the `shmat()` system call to attach shared memory.
- Uses the `gang_enter` system call to enter the gang and executes the `block()` system call.
- Reports to the parent process through the shared memory location.

After the child process reports, the parent process performs the following operations:

- Starts the timer, executes the `_unblockF` system call to unblock the child process, stops the timer and reports the execution time for `_unblockF` system calls in the following format:

```
x seconds/_unblockF
```

Variable	Description
x	Specifies the execution time for the `_unblockF` system call.

- Enables the child writing output, exits the gang, and waits for the child to complete execution.

The child process then performs the following operations:

- Reads from memory. Execution times are reported in the following format:

```
x seconds/RD time
```

Variable	Description
x	Specifies the execution time for a read operation.

- Writes to memory. Execution times are reported in the following format:

```
x seconds/RW time
```

Variable	Description
x	Specifies the execution time for a write operation.

shmtest

The `shmtest` routine verifies the operation of the `shmpublish()` and `shmreplace()` system calls. The `shmpublish()` call converts a segment of memory into shared memory and the `shmreplace()` call replaces a memory segment with a shared memory segment. This task verifies that the correct memory is exported and overridden and invalid operations are detected. The parent process in this routine performs the following operations:

- Verifies that shared memory is configured correctly and reports the following information:

```
shmtest: Offset o shared x
```

Variable	Description
o	Specifies the offset of the shared data segment.
x	Specifies the address of the shared data segment.

- Attempts to convert its `.text` region into a shared memory segment with the `shmpublish()` system call. The task reports the following milestone prior to starting the operation:

```
shmtest.id.PAR: shmpublish text
```

Variable	Description
id	Specifies the process identification number for the parent process.

This operation should fail because the `.text` portion of a process cannot be published.

- Attempts to convert its stack into a shared memory segment with the `shmpublish()` system call. The process reports the following milestone prior to starting the operation:

```
shmtest.id.PAR: shmpublish stack
```

Variable	Description
id	Specifies the process identification number.

This operation should fail because the stack portion of a process cannot be published.

- Uses the `shmpublish()` system call to publish the existing shared data segment. The task reports the following milestone prior to executing the system call:

```
shmtest.id.PAR shmpublish(PRIV, x, DEST+CREAT+0777)
```

Variable	Description
id	Specifies the process identification number.
x	Specifies the address of the shared data segment.

This operation should pass because segments from `.shrdata` in an executable file can be published.

- Attempts to replace the published region over itself. The task reports the following milestone prior to executing the `shmreplace()` system call:

```
shmtest.id.PAR: shmreplace(x, self, RND)  (failok)
```

Variable	Description
id	Specifies the identification number of the parent process.
x	Specifies the shared memory segment identification number.

This operation should fail because replacing a published shared memory segment with itself is not a valid operation.

- Attempts to replace the published region somewhere else in system memory. The task reports the following milestone prior to executing the `shmreplace()` system call:

```
shmtest.id.PAR: shmreplace(x,0,0)
```

Variable	Description
id	Specifies the identification number of the parent process.
x	Specifies the shared memory segment identification number.

This operation should be successful.

- Verifies the data in the new memory segment. The task reports the following milestone:

```
shmtest.id.PAR: attached id x(y) at z
```

Variable	Description
id	Specifies the identification number of the parent process.
x	Specifies the identification number of the shared memory segment in decimal format.
y	Specifies the identification number of the shared memory segment in hexadecimal format.
z	Specifies the location of the shared memory segment.

- Verifies that shared memory is still valid after the attach.
- Builds a child process with the `fork()` system call and waits for the child to execute. The parent reports the following milestone while waiting for the child:

```
shmtest.id:Barrier2 ...
```

Variable	Description
id	Specifies the identification number of the parent process.

While the parent process is suspended, the child process performs the following operations:

- Uses the `execv()` system call to obtain a new `.shrdata` region. The child process reports the following milestone prior to executing the system call:

```
shmtest id2: execv(shmtest, argv[shmtest, -X, x,-p, 0], 0)
```

Variable	Description
id2	Specifies the identification number of the child process.
x	Specifies the identification number of the memory segment.

- Verifies its shared memory is configured correctly and reports the following milestone:

```
shmtest: Offset o shared x
```

Variable	Description
o	Specifies the offset of the shared data segment.
x	Specifies the address of the shared data segment.

- Replaces its .shrdata region with the region published by the parent process. The child reports the following milestone:

```
Exec.shmtest.id2: shmreplace
```

Variable	Description
id2	Specifies the identification number of the child process.

- Verifies the contents of the newly replaced area. The following report is issued before the verification starts:

```
Exec.shmtest.id2: Verify
```

Variable	Description
id2	Specifies the identification number of the child process.

- The child process reports status to the parent process. The parent process waits for a response from the child and reports the following milestone:

```
shmtest.id: Barrier2 done, Wait
```

Variable	Description
id	Specifies the parent process identification number.

The parent process reports the following milestone when the child process completes execution:

```
Child Done
```

block_test2

The `block_test2` routine executes the `unblock(getpid())` system call followed by the `block()` call. If the test executes successfully, the `block()` call becomes a no operation command. The task prints out the following information if execution completes successfully:

```
It is an error if this sentence isn't underlined.
================================================
```

If the task hangs before completing the second line of the message, an error has occurred. Errors are also indicated with error messages.

Error Messages

If an error occurs during UMAX V Exerciser operation, one of the following types of errors is reported:

- Timing
- Process Control
- Child Process
- Cache Control
- Shared Memory Control

Timing Error Messages

If a task determines that its timing loop takes too long to execute, the exerciser displays one of the following messages:

```
Panic: seconds/op > 9
Panic: seconds > 9
Panic: mseconds/op > 999999
Panic: mseconds > 999999
```

If a loop overhead problem is detected in the `tproc` routine, the exerciser displays:

```
Panic: tv_usec >= 1000000
```

If timing errors are detected in the timing loop used by the `block_test2` routine, the exerciser displays one of the following messages:

```
block_test2: timed_block SUCCESS ERROR-(bad addr)
block_test2: timed_block SUCCESS ERROR-(sec<0)
block_test2: timed_block SUCCESS ERROR-(usec<0)
```

Process Control Error Messages

Process control error messages may be produced by one of the following exerciser routines:

- `plock_test`
- `plock_test2`
- `block_test2`
- `tproc`
- `tyld`

These errors are associated with one of the following process control operations:

- Process lock/unlock - `plock()`
- Process block/unblock - `block()`, `unblock()`
- Make a process memory resident - `plockres()`
- Create a new process - `fork()`
- Create a set of processes - `gang()`

`plock()` Error Messages

If an error is detected while testing the `plock()` system call in the `plock_test` routine, the exerciser displays one of the following error messages:

- If the `plock()` call works when the task does not have superuser privileges:

```
ERROR: plock worked non-root
ERROR: punlock worked non-root and non-locked
```

- If the task fails to execute a `TXTLOCK` or `DATLOCK` command when no other locks are present:

```
ERROR: plock(txtlock) fails
ERROR: plock(datlock) fails
```

- If the task successfully executes a `TXTLOCK` or `DATLOCK` command when other locks are present:

```
ERROR: plock(txtlock) after proclock
ERROR: plock(datlock) after proclock
```

- If the task successfully executes a PROCLOCK command when a text or data lock is already present:

```
ERROR: plock(proclock) after txtlock
ERROR: plock(proclock) after datlock
```

- If the PROCLOCK command fails when no other locks are present:

```
ERROR: plock(proclock) fails
```

- If the UNLOCK command fails:

```
ERROR: plock(unlock) fails
ERROR: plock(unlock) after unlock
```

block(), unblock() Error Messages

If the process is unable to unblock itself using the unblock() system call during the execution of the block_test2 routine, the exerciser displays:

```
block_test2 pidn: unblock(self) ERROR
```

Variable	Description
n	Specifies the process identification number.

If the block() system call returns the wrong value to the calling process during the execution of the block_test2 routine, the exerciser displays:

```
block_test2: ERROR return value NOT ZERO
```

`plockres()` Error Messages

If the memory access times for pages before and after the execution of the `plockres()` system call are not significantly different when the `plock_test2` routine is executed, the exerciser displays:

```
ERROR: NOT ENOUGH FASTER after plockres(2)
```

`fork()` Error Messages

If the child process produced with the `fork()` system call during the `tproc` routine has an exit failure, the exerciser displays:

```
Panic: Exit failure
```

If no child was created when the `fork()` call was executed in the `tproc` routine, the exerciser displays:

```
wait error - no child
```

`gang()` Error Messages

If the `tyld` routine is unable to create a gang, the exerciser displays:

```
Max gang size of x
```

Variable	Description
x	Specifies the value returned by `gang_maxsize`.

If there's a problem with the gang set up in the `tyld` routine, the exerciser displays:

```
Gang setup error
```

Child Process Error Messages

If the child process returns incorrect status to the parent process during the execution of the `async_test` routine, the exerciser displays:

```
Bad Kid Wait Status
```

If an error occurs in the child process while testing the `CCTL_RONLY` command in the `cctl_test2` routine, the exerciser displays one of the following messages:

- If a read error occurs:

```
Child: Bad Pattern
```

- If the child is able to write to read only memory:

```
Child ERROR: Wrote RO Memory and LIVED!
```

- If there is no response from the child process:

```
Parent: RO Child ERROR
```

If a child process error occurs during the execution of the `shmtest` routine, the exerciser displays one of the following messages:

- If the child process cannot execute the `execv()` system call:

```
Forked child: execv FAILED
```

- If the `execv()` system call fails:

```
Exec.shmtest.id: Init error a[x] = y
```

Variable	Description
id	Specifies the child process identification number.
a	Specifies the address of the shared data segment.
x	Specifies the shared memory location.
y	Specifies the data read from the location.

Cache Control Error Messages

If a speed error occurs when testing the `cachectl()` system call in the `cctl_test` routine, the exerciser displays one of the following messages:

```
ERROR: Disabled cache is faster
ERROR: Disabled cache is Not much slower
```

If using the `CCTL_PLOCK` command in the `cctl_test3` routine did not speed up the memory access time, the exerciser displays:

```
ERROR: NOT ENOUGH FASTER after cachectl(2)
```

Shared Memory Control Error Messages

The following error messages are associated with failures in the `shmtest` routine:

- If the shared memory used by the routine has incorrect data:

```
shmtest ERROR: share_data[x] = 0xy
```

Variable	Description
x	Specifies the shared memory location.
y	Specifies the data from the location.

- If the uninitialized portion of the shared memory used by the routine contains incorrect data:

```
shmtest ERROR: shrbss[0,1] = 0xa, b
```

Variable	Description
a	Specifies the data in uninitialized location 0.
b	Specifies the data in uninitialized location 1.

- If the parent process is able to publish its `.text` region:

```
Parent ERROR: Successfully published .text
```

- If the parent process is able to publish its stack region:

```
Parent ERROR: Successfully published stack.
```

- If the parent process is able to replace the published area over itself:

```
shmtest.id ERROR: shmreplace(x,y,RND) SHOULD FAIL
```

Variable	Description
id	Specifies the process identification number.
x	Specifies the shared memory segment identification number.
y	Specifies the shared memory segment address.

CHAPTER 10

Generic Exerciser

The Generic Exerciser allows you to incorporate tests into the Online Exerciser. This exerciser executes a user-defined shell command and returns the standard output and standard error lines in the message format defined by the Online Exerciser.

Special Requirements

There are no special requirements associated with running the Generic Exerciser. However, the user-supplied commands and test scripts may have special requirements.

Unique Parameters

In addition to the parameters described in Chapter 3, the Generic Exerciser provides the following unique test parameter.

TABLE 10-1 Generic Exerciser Parameters

Parameter	Description
Command	Enter the shell command to be executed. This command is executed by the shell in the following format: `sh -c` *command* *command* specifies the value given to this parameter. The default command is `echo Milestone` which echoes the word "Milestone" to the standard output. The *command* is executed in the `/usr/oe/test/generic` directory. If the command specified is not found in the `PATH` environment variable and is not a valid file name relative to that directory, an error will occur. Avoid problems by entering the full path name when specifying commands.

Command Examples

This section provides examples using the Generic Exerciser to test a variety of system components. In each example, the command specified represents the value given to the Command parameter.

Tape Drive Test

The following command tests tape drive `0`:

```
/bin/dd if=/dev/dsk/0s2 of=/dev/rmt/0h 2>&1
```

Disk Drive Test

The following command performs reads of disk drive `0s`:

```
/bin/dd if=/dev/rdsk/0s2 of=/dev/null 2>&1
```

Network Test

The following command is used to test connectivity to remote host `alien`:

```
/usr/bin/ping alien 0 1
```

User-Supplied Test Script

The following command is used to execute the user-supplied test script `/user0/guest/doit`:

```
/user0/guest/doit
```

Note – The user-supplied test script must exist and must have execute permission.

Messages

This section provides some general information about messages generated during the execution of the Generic Exerciser. Two types of messages are associated with this exerciser:

- Milestone
- Error

Milestone Messages

Each line of standard output from the shell command becomes a milestone message when the exerciser is running. However, not all output will be logged unless the Milestone Statement parameter is set to `All Milestones`.

The following milestone messages are associated with a child process:

```
Child terminated on signal x stat
Child exited normally (Status stat)
Child stopped on signal x
```

Variable	Description
x	Specifies the signal causing the child process to terminate or stop.
stat	Specifies the status of the child process. For a terminated child, the status indicates if core was dumped. For a normal exit, the status is the lower 8 bits of the returned exit status.

Error Messages

Each line of standard error from the execution of the shell command becomes an error message when the exerciser is running. The most common error messages fall into one of the following categories:

- System Call errors
- Shell Errors

If an error is detected, the program displays a specific error message and one of the following messages:

```
Generic Exerciser failed
Generic Test Failed
```

System Call Error Messages

System Call error messages are associated with the failure of specific system calls like `fork()` and `exec()`. These errors are not typically user-related.

System Call error messages are expressed in the following format:

message`:` *errorname* `(Error` *n*`)`

Variable	Description
message	Specifies a test defined message related to the error. The test defined messages are displayed in the following format: `Error during` *call* *call* Specifies the failing system call.
errorname	Specifies a system error message associated with the failing system call. System error messages are described in `intro(2)` in the *UMAX V Programmer's Reference* manual.
n	Specifies the error number associated with a system error message.

Shell Error Messages

Shell error messages are generated by the shell prior to command execution. If an attempt is made to execute a program that does not exist or cannot be located, the shell generates the following error message:

```
sh: prog: not found
```

If an attempt is made to execute a program that does not have execute permission, the shell generates the following error message:

```
sh: prog: cannot execute
```

Variable	Description
prog	Specifies the name of the program specified in the `Command` parameter.

APPENDIX **A**

Man Pages

The following pages show the man page information for the Online Exerciser. This information is displayed when you enter **man oe** at the UMAX V prompt.

NAME

oe – program to run online diagnostics

DESCRIPTION

The Online Exerciser Program (oe) is a system exerciser that runs under the UMAX V operating system. It tests both hardware and software under normal system operating conditions.

SYNOPSIS

oe [**-cdv**] [**-l***arg*] [**-m***arg*] [**-o***arg*] [**-q***arg*] [**-r***arg*] [**-t***arg*] [*testgroupname*]

OPTIONS

One or more of the following options can be entered as part of the command line:

Option	Description
-c	Determines and displays the current configuration of all the tests on the local and remote systems as defined in the `remotelib` and `testlib` files. The exerciser does not run any test groups, nor does it enter the group editor, when this option is selected.
-d	Enables the generation of debug messages.
-l*arg*	Overrides the default path for the library files (`remotelib` and `testlib`) and the test directory. The default directory path is `/usr/oe`. Enter the complete pathname.
-m*arg*	Selects the Online Exerciser operation mode. The following are valid arguments: **b** Batch Mode. This mode allows testgroups to be run from the command line. **e** User Mode. This mode provides the user interface. You can create / modify test groups and run test groups while running in this mode. This is the default if the **-v**, **-c**, and **-mb** options are not selected.
-o*arg*	Overrides the log file named in a test group. If *arg* is a **-**, the log output is directed to `stdout`.
-q*arg*	Overrides the test directory path relative to the library path. For example, under `/usr/oe` instead of `test` being the test directory, another directory such as `alpha` could exist. You would type the option selection **-qalpha**. The path for the new test directory would be `/usr/oe/alpha`.
-r*arg*	Reads the remote library from a file other than `remotelib`. The only valid argument is a new file name. The file must be in the library directory. The default is `/usr/oe/remotelib`.
-t*arg*	Reads the test library from a file other than `testlib`. The only valid argument is a new file name. The file must be in the library directory. The default is `/usr/oe/testlib`.
-v	Displays information about the current version of the Online Exerciser.

DESCRIPTION

The Online Exerciser Program (`oe`) is a system exerciser that runs under the UMAX V operating system. It tests both hardware and software under normal system operating conditions.

EXAMPLES

Command line execution of a test group where `mem.oeg` is a test group:

```
./oe -mb mem.oeg
```

To bring up the user interface of the Online Exerciser program enter

```
./oe
```

SPECIAL CONSIDERATIONS

- You must have superuser privileges to run this program.
- If you want to run tests over the network with this program, the name of the remote system must be listed in `/etc/hosts`, `/.rhosts`, and `/etc/hosts.equiv` on the local system. The remote system must be a trusted host.

SEE ALSO

Sun StorEdge A7000 Online Exerciser Reference Manual