Sun StorEdge™ A7000 Online Exerciser Programmer's Guide

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. 805-6661-10
January 1999, Revision A

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

Please
Recycle

Contents

Figures and Code Samples

Tables

Preface

Sun StorEdge A7000 Online Exerciser Programmer's Guide describes how to write tests to be executed under the control of the Online Exerciser (OE) program. The Online Exerciser is a system exerciser designed to run under the operating system on a Sun StorEdge A7000 Intelligent Storage Server system. This manual contains the following information:

- An overview of the Online Exerciser operating environment.
- A description of the operations performed during the start up of the Online Exerciser.
- Descriptions of the structures and functions required in Online Exerciser test programs.
- Descriptions of the library routines used to interface the test program to the Online Exerciser.
- An Online Exerciser test program example.

How This Book Is Organized

Chapter 1 "Online Exerciser Environment" describes the relationship between the user, the Online Exerciser program, and the subsystem under test. The information in this chapter includes:

- Descriptions of the Online Exerciser control and test processes and their relationships with the user and the subsystem under test.
- A description of the communications between Online Exerciser processes.
- A description of the operations performed when the Online Exerciser starts.

Chapter 2 "Test Program Requirements" describes the individual functions and structures that must be present in any test program run under the control of the Online Exerciser. It also provides descriptions of the types of messages that can be sent by the test program to the Online Exerciser control program to be either logged or displayed.

Chapter 3 "Library Routines" describes the individual functions provided in the Online Exerciser library (`liboe.a`) for use by the test program. These functions are used to report messages, pass parameters, and execute code residing outside the test program.

Chapter 4 "Sample Test Program" describes a sample bare minimum test program that can be used by the test developer as a training aid when learning to build and execute user-developed test programs under the control of the Online Exerciser. This chapter includes procedures for:

- Generating executable images.
- Adding a test program to the Online Exerciser file structure.
- Executing a user-developed test program.

Typographic Conventions

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output.	The Online Exerciser library (`liboe.a`) contains individual functions used by the test program. `Determining Local and Remote Configuration`
`AaBbCc123`	What you type, when contrasted with on-screen computer output.	**`./oe -mb`**
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 Reference Manual.* *hostname* `Exerciser Template devel template`
. . .	The horizontal ellipsis indicates repetition or omission.	In the following example, one or more groups can be entered: **`oe -mb`** [*options*] *group1* [*group2*, . . .]

Note – The pathnames used in this document are the default pathnames established when the Online Exerciser was installed.

Related Documentation

TABLE P-2 Related Documentation

Type	Title
Diagnostic reference	*Sun StorEdge A7000 Online Exerciser Reference Manual*

Sun Documentation on the Web

The `docs.sun.com`[sm] 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 Environment

The Online Exerciser is divided into two main functions: the control process and the test process. FIGURE 1-1 shows the relationship between these processes, the user, and the system element under test.

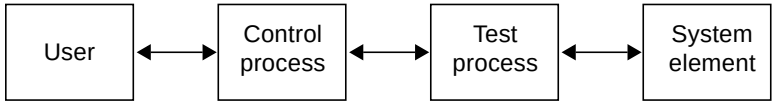

FIGURE 1-1 Process Relationships

The control process interfaces between the user and the test process. It provides a mechanism for performing the following operations:

- Determining the system configuration
- Interacting in a friendly way with the user
- Running a suite of tests
- Collecting the output data resulting from test execution

The test process receives and processes commands and user-defined parameters from the control process. It also returns data collected from running its various routines and test code to the control process. The test process contains:

- Configuration routines
- Initialization routines
- Cleanup routines
- System element specific test code

Note – Before adding test programs to the Online Exerciser, you must become familiar with the Online Exerciser file system structure described in the *Sun StorEdge A7000 Online Exerciser Reference Manual*.

The Online Exerciser control program sees only the contents of the `testlib` and `remotelib` files. When adding a new test to the Online Exerciser, ensure that the test is listed in the appropriate files.

Control Process

The control portion of the Online Exerciser is the `oe` program located, by default, in the `/usr/oe` directory. It contains the following functions:

- Configuration manager
- Group editor
- Test controller

The configuration manager obtains and stores the configuration or set of parameters for each test.

The group editor provides a user interface for examining and modifying test groups, the test library, and the remote library. It also allows you to execute test groups and rerun the configuration manager.

Note – The configuration manager should be rerun if you modify either the test library or the remote library.

The test controller is used to run a test group. It is activated either by selecting the `Execute` option from the user interface menu or by using the `-mb` argument to execute the `oe` program.

Test Process

The test process contains a test executive program and subsystem specific test code. The test directory contains subdirectories of tests and related files used for testing individual subsystem elements. By default, the test directory is `/usr/oe/test`. The subdirectory `/usr/oe/test/memory`, for example, contains memory tests and related files.

The test executive manages the communications between the test code and the control process. It also provides a simple executive to iterate the functions of the test. The test executive resides in the Online Exerciser library (`liboe.a`) and provides the main routine (`main()`) for the test code.

The subsystem specific test code contains a minimum set of functions to configure and test a specific part of the system. The set of functions is defined by the `ose_test` structure in the Online Exerciser include file (`oe.h`). These functions include:

- `config()`
- `param()`
- `init()`
- `cleanup()`
- At least one test function

The test structure is described in Chapter 2.

The Online Exerciser file system structure is described in the *Sun StorEdge A7000 Online Exerciser Reference Manual*.

Process Communications

The Online Exerciser program uses the socket interface for communications between the test process and the control process.

The control process performs the following operations:

1. Uses the ARPA Internet Address format and IP protocol to request a streams socket. This type of socket interface provides sequenced, reliable, two-way, connection based byte streams.
2. Executes the `bind` command to request binding a name to the socket.
3. Sets the socket to listen mode.
4. Starts the test executive in the appropriate test directory.
5. Passes its socket port number and the host name to the test executive as command line arguments.

The test executive then performs the following operations:

- Requests a streams socket using the ARPA Internet Address format and IP protocol. This socket is connected to a specific host.
- Attempts to establish a connection to the socket used by the control process.

- Once the control process accepts the connection, the test executive is ready to receive commands from and return data and status to the control process.
- The connection between the control process and the test process is terminated if either a configuration request is satisfied or a `shutdown` command is sent to the test process by the control process.

Startup Operations

The Online Exerciser can run in several different modes. The operation mode is determined by the command line argument used to start the Online Exerciser. The following run arguments are available:

Argument	Mode
`-v`	Version
`-c`	Configuration
`-mb`	Batch
`-me`	User Interface

If none of these modes are selected, the program defaults to user interface mode (`-me`).

Additional command line arguments are provided for performing other operations. For example, **`./oe -l /tmp`**, indicates that the Online Exerciser test directory is in `/tmp`. The command line arguments are described in the *Sun StorEdge A7000 Online Exerciser Reference Manual*.

Type **`./oe`** and the desired options to start the Online Exerciser.

Once started, the Online Exerciser performs the following operations:

1. Determines whether any command line arguments were entered. If the version mode (`-v`) was specified, the program displays the version information and exits immediately. If no argument or the `-me` argument was specified, the program displays the following message:

```
Determining Local and Remote Configuration

Please Wait
```

   This message is bypassed for the other run mode arguments (`-v`, `-c`, `-mb`).

2. Initializes some global variables to their default values. The default values include running the exerciser tests in parallel mode and storing all messages (milestone, error, case, and pass) in the log file. The program also obtains the local host name.

3. Once initialization is complete, the Online Exerciser verifies that the user has superuser privileges. If not, the program displays the following message and exits:

```
You must be superuser to run the Online Exerciser.
```

4. If superuser privileges are enabled, the Online Exerciser then checks the path to the test directory. The default path is `/usr/oe`. The program next verifies that the test directory exists. The default directory name is `test` and the default path is `/usr/oe/test`. Finally, the Online Exerciser verifies that it can open the `testlib` and `remotelib` files. These files are typically located in `/usr/oe`. If any errors are encountered while performing this confidence check, the program posts an error message and exits.

Note – The `remotelib` and `testlib` files can be empty, but they must exist. If both of these files are empty, nothing is available to test.

---

5. If the Online Exerciser is running in configuration mode (`-c`), it determines and displays the current configuration of all the tests on the local and remote nodes, as defined in the `remotelib` and `testlib` files, and then exits.

   If you are running in batch mode (`-mb`), the Online Exerciser program determines whether a test group was specified. If no test group or an incorrect test group was specified, the program displays the following error message and exits:

```
Usage: oe -mb [options] group1 [group2,...]
```

At this time, if the Online Exerciser is running in either user interface or batch mode, the configuration manager is activated to obtain the complete configuration for all local and remote nodes based on the information in the remote library and test library. The configuration manager performs the following operations:

1. Using a semaphore, protects the `testlib` and `remotelib` files, allowing multiple copies of the Online Exerciser to run simultaneously without fear of a collision when these files are accessed.

2. Connects to each test executive through a sockets interface.

3. Once the connection is established, runs the configuration function found in each test executive and saves the configuration information. This operation is performed for each test program listed in the `testlib` file of each selected host (local and remote).

Note – Typically, each test directory contains one test executive. For example, `rmstest` in `/usr/oe/test/rms` contains the test executive used for interfacing with the control process.

When the configuration manager is finished, the Online Exerciser either comes up in user interface mode or starts executing the test group specified with the `-mb` argument. In other words, either the group editor takes control to provide the user interface or the test controller takes control to run the tests.

CHAPTER 2

Test Program Requirements

You can write new test programs or port existing programs to run under the control of the Online Exerciser if the programs meet the necessary requirements for interfacing with the Online Exerciser control program.

The Online Exerciser control program expects the test programs to use certain structures defined in the include file (oe.h). It also requires test programs to provide certain functions to be activated by the control program.

Library routines are provided in `liboe.a` to facilitate interfacing new test programs with the Online Exerciser. In particular, the library routines enable the control program to differentiate between the types of messages returned by a test program. Refer to Chapter 3 for a description of each user callable library function.

Note – A bare minimum sample test program is provided with the Online Exerciser software. Refer to Chapter 4 for additional information.

Required Structures

The following structures are required in any Online Exerciser test program:

- Result structure (`t_result`)
- Test structure (`ose_test`)

Result Structure

The result structure (`t_result`) provides the format necessary to pass the results of the required test functions to the Online Exerciser control program. Each required test function described in “Required Functions” returns a pointer to this structure. In the following example, each function specified returns a pointer to a structure of type `t_result`:

```
struct t_result *RMS_Config(), *RMS_Param(), *RMS_Init(),
*RMS_Cleanup(), *RMS_Test();
```

The `t_result` structure contains the following elements:

- Result code (`r_code`)
- String pointer (`*r_message`)

Two result codes are defined in `oe.h`: `PASSED` and `FAILED`.

A function returns a result code of `PASSED` if the function executed normally. In this case the string pointer (`*r_message`) is ignored. It is recommended, however, that a known value be placed in the string pointer anyway. For example,

```
r_code=PASSED;
*r_message=NULL;
```

where `NULL` is defined in `<stdio.h>`.

A function returns a result code of `FAILED` if the function did not execute normally. In this case the string pointer (`*r_message`) points to the reason for the failure. For example,

```
r_code=FAILED;
*r_message=”Unable to open remotelib”;
```

Note – A newline is not required for this string because the message will be formatted by the Online Exerciser control program.

Test Structure

The test structure (`ose_test`) provides the format for the required functions and any additional test functions. This structure is used to build an array of functions. The array is used by the Online Exerciser control program to access the various functions in the test program. The functions of a test program must be listed in the test array in a certain order.

Each entry built under the `ose_test` structure contains three elements:

- Function pointer
- String pointer
- Flags

The function pointer provides the entry point into the test code for the Online Exerciser control program. Notice that the test name entry in CODE EXAMPLE 2-1 contains a `NULL` function pointer because no function is associated with this entry. The last entry in the array must also always contain a `NULL` function pointer.

The string pointer indicates the purpose of the function. For example, "`RMS Configuration`" describes the operations performed by function `RMS_Config`. The string pointer of the last entry is always `NULL`.

The flags are `NULL` for all functions except test functions. For test functions, the flags are either `TEST_ENABLE` or `TEST_SKIP`. The value selected is the desired default setting for a specific test. In CODE EXAMPLE 2-1, the test function `RMS_Test` is set to `TEST_ENABLE` by default. You can change the default setting with the `Param` option provided by the user interface.

The Online Exerciser include file, `oe.h`, defines the order of the first five array elements as follows:

```
#define TEST_NAME 0
#define TEST_CONFIG 1
#define TEST_PARAM 2
#define TEST_INIT 3
#define TEST_CLEAN 4
#define TEST_FIRST 5
```

In CODE EXAMPLE 2-1, the functions are listed in their required order:

CODE EXAMPLE 2-1 Test Structure Functions

```
struct ose_test test[]={
NULL, “RMS Exerciser”, NULL,
RMS_Config, “RMS Configuration”, NULL,
RMS_Param, “RMS PArameters”, NULL,
RMS_Init, “RMS Initialization”, NULL,
RMS_Cleanup, “RMS Cleanup”, NULL,
RMS_Test, “RMS Test”, TEST_ENABLE,
NULL, NULL, NULL};
```

Required Functions

Each Online Exerciser test program must contain a minimum of five functions. The following functions are required:

- Configuration
- Set Parameters
- Initialization
- Cleanup
- One or more test functions

Any of the functions may be stubs, but each function must exist and must return a pointer to a structure of type `t_result`.

The following functions are called once for each pass of the Online Exerciser task:

- The initialization function, which prepares the Online Exerciser task to run.
- The cleanup function, which performs any Online Exerciser task cleanup required before task completion.
- A least one test function.

The configuration function is called only once when the Online Exerciser control program is started or when you select the `Config` option from the user interface menu. This function returns a list of configuration variables to the control program.

The set parameters function is called during the Online Exerciser test group initialization process and is used to set any exerciser variable that is not to be left in its default state. The function is called once for each variable to be set.

Configuration Function

The configuration function returns a list of variables to the control program. These are variables that you can modify. This list could include the devices available for testing, the size of a buffer to be tested, or some other value. Variables are expressed in two different forms: discrete parameters and string parameters.

Discrete parameters can be set only to certain predetermined values. For example, only configured devices can be exercised with the disk drive exerciser. In this case, the configuration function returns a list of the currently configured drives and the user selects the drives to be tested from this list. The default value is normally the first item of that type returned to the control program.

String parameters can be set to an unspecified range or name. For example, the disk drive exerciser defines the blocksize to be tested as a string parameter. The configuration function returns a maximum string length and a default value for this parameter.

CODE EXAMPLE 2-2 displays an example of some of the configuration code used for the memory exerciser. Notice that the Run Test Ganged parameter is a discrete parameter. This parameter can be enabled and disabled only by the user. In contrast, the Fill Pattern parameter is a string parameter. The user can enter the desired pattern up to 16 characters.

Each parameter displayed in CODE EXAMPLE 2-2 has an associated code. The code is defined as an offset of the `TEST_UNIQUE` value defined in `oe.h`. Code values between 0 and `TEST_UNIQUE-1` are reserved for the Online Exerciser control program. The test specific codes are used by the Online Exerciser control program when it calls the set parameters (`param()`) function of the test program.

CODE EXAMPLE 2-2 may be easier to understand if you bring up the Online Exerciser program. Enter **`./oe`** in the `/usr/oe` directory to bring up the Online Exerciser in interactive mode. Select the `Group` option followed by the `Add` option. Then select the Memory Exerciser followed by the `Param` option.

Note – Functions `ose_out_disc_param()` and `ose_out_string_param()` are described in Chapter 3. These functions are available in `liboe.a`.

CODE EXAMPLE 2-2 Configuration Function Example

```
/* Configuration Function Example */

#define Code_Fill_Pattern    TEST_UNIQUE
static char Fill_Pattern[17];

#define Code_Gang_Enable     TEST_UNIQUE+1
static int Gang_Enable;

struct t_result *test_config()
{
static struct t_result result;
char Buff[100];

/*Specify fill pattern */
strcpy(Fill_Pattern,"0123456789abcdef");
ose_out_string_param(Code_Fill_Pattern,"Fill Pattern",16,Fill_Pattern);

/*Specify ganging params*/
if{Gang_Enable)
   {
   ose_out_disc_param(Code_Gang_Enable,"Run Test Ganged",1,"Enable");
   ose_out_disc_param(Code_Gang_Enable,"Run Test Ganged",0,"Disable");
   }
else
   {
ose_out_disc_param(Code_Gang_Enable,"Run Test Ganged",0,"Disable");
   ose_out_disc_param(Code_Gang_Enable,"Run Test Ganged",1,"Enable");
   }

result.r_code = PASSED;
result.r_message = NULL;
return(&result);
}
```

Set Parameters Function

The set parameters function is called by the Online Exerciser control program when a configuration variable is to be set to a value. CODE EXAMPLE 2-3 displays an example of the code the control program expects to call based on the configuration function example displayed in CODE EXAMPLE 2-2.

CODE EXAMPLE 2-3 Set Parameters Function Example

```
/* set parameter */
struct t_result *test_param(code,selection,strptr)
int code;
int selection;
char *strptr;
{
static struct t_result result;
switch(code)
   {
    case Code_Fill_Pattern:
    strcpy(Fill_Pattern,strptr);
    break;

    case Code_Gang_Enable:
    Gang_Enable = selection;
    break;
   }
result.r_code = PASSED;
result.r_message = NULL;
return(&result);
}
```

The first argument, `code`, contains the ID of the parameter to be modified.

The second argument, `selection`, is an integer indicating which value of a discrete parameter has been selected. This argument is used only to modify discrete parameters.

The third argument, `strptr`, contains a pointer to an ASCII string containing the value to be used by a string parameter.

Initialization Function

The initialization function contains initialization code specific to a test program that must be executed before the start of each pass of an exerciser test. This function is called once per pass of the test. The memory exerciser uses this routine to request the memory space to be tested. If a minimum of Mandatory and Major Milestones is enabled, the memory exerciser also posts a milestone message in the following format while in its initialization routine:

```
"Allocated n continuous bytes of memory starting at 0xaaaaaaaa"
```

Variable	Description
n	Indicates the number of bytes of memory to be tested.
aaaaaaaa	Indicates the physical starting address of the memory area.

Cleanup Function

The cleanup function contains cleanup code that is specific to the test program and must be executed at the end of each pass of the test program. For example, the memory exerciser test program frees the memory space allocated during the initialization routine.

Test Function

The test functions contain subsystem test code. A test program can have one or more test functions. A test function can either contain the test code or execute code outside the test function.

The `memtest` executable located in `/usr/oe/test/memory` contains 13 separate test functions as well as the required `config()`, `param()`, `init()`, `cleanup()`, and test executive `main()` functions. Each of these test functions, such as Sequential Fill Test and Increment Test, is displayed in the `Param` menu and can be configured to either the Enable or Skip state.

In the network exerciser directory, `/usr/oe/test/network`, are two executables: `rcptest` and `nettest`. The `nettest` executable contains the required Online Exerciser functions as well as the test function that calls `rcptest`. `rcptest` is a script file that can be run either standalone or under the control of `nettest`. Any code external to the test executive can be run in this manner. CODE EXAMPLE 2-4 displays the test function for the network exerciser.

CODE EXAMPLE 2-4 Network Exerciser Test Function

```
/* Example of executing external test code */

char host[256];
struct t_result *test_1()
{
static struct t_result result;
char Cmd[100];

sprintf(Cmd,”./rcptest %s”,host);
Execute_Test(Cmd);

result.r_code = PASSED;
result.r_message = NULL;
return(&result);
}
```

Note – The `Execute_Test()` function is described in Chapter 3. This function is available in `liboe.a`.

Program Messages

The Online Exerciser control program logs the following types of messages to either a file or `stdout`:

- Test Started
- Test Finished
- Case Completed
- Pass Completed
- Error Milestone
- Debug
- Abort

The test code sends Error, Milestone, Debug, and Abort messages to the control program for posting. All other types of messages are status messages posted by the control program.

Error Messages

Error messages are posted when they are received by the control program from either `stderr` or the test program. The test program sends an error message by using the `ose_out_error` routine available in `liboe.a`. This routine is described in Chapter 3.

Milestone Messages

Milestone messages are posted when the control program receives messages from either `stdout` or the test program. The test program uses the `ose_out_milestone` routine in `liboe.a` to send milestone messages to the control program. The `ose_out_milestone` routine allows the test program to provide multiple levels of milestones. The level of output is specified when a test group is created. The milestone reporting routine is described in Chapter 3.

Debug Messages

Debug messages are posted by the control program only when messages sent with the `ose_out_debug` routine are received from the test program. This routine is also located in `liboe.a`. Like milestone messages, debug messages also provide multiple levels of output. The level of output is specified when a test group is created. The default level is `No Debug Messages` because this type of message is primarily used by the test code developer for code debug. The debug message reporting routine is also described in Chapter 3.

Abort Messages

Abort messages are posted by the control program to indicate that the test code is not completing in a normal manner. The test program uses the `ose_out_abort_notice` routine in `liboe.a` to send this type of message to the control program. This message routine is described in Chapter 3.

CHAPTER 3

Library Routines

Library routines are provided in `liboe.a` to facilitate interfacing user-developed test programs with the Online Exerciser control program. These routines are grouped into the following categories:

- Message routines
- Configuration variable passing routines
- `Execute_Test()` routine
- `main()` routine

Message Routines

Routines provided in `liboe.a` enable a test program to send data to the operator through the Online Exerciser control program. The following types of message routines are available:

- Debug
- Milestone
- Error
- Abort

These routines are valid for functions called directly by the Online Exerciser control program. If a test program uses the `Execute_Test()` function to execute test code external to the test program, the external code must be written to post milestone messages to `stdout` and error messages to `stderr`. The message routines (`ose_out_debug`, `ose_out_error`, `ose_out_milestone`, and `ose_out_abort`) cannot be used in the external test code.

Debug Message Routine

Use the ose_out_debug() function to send data useful for debugging a test program run under the control of the Online Exerciser. This function has the following format:

```
ose_out_debug(message,level);
char *message;
int level;
```

The first argument, message, is a pointer to a string containing the debug message to be displayed. Newline characters are not required at the end of the string because the Online Exerciser control program formats the string into a statement containing the test program ID, time, and other useful information.

The second argument, level, is an integer from -1 to 2. This value represents the relative importance of the message. Four options for the Debug Statements parameter are available from the Param menu. Each option is assigned a value as follows:

Value	Option
0	No Debug Statements
1	Major Debug Statements
2	Most Debug Statements
3	All Debug Statements

When a message is passed to the Online Exerciser, the control program compares the message level with the value of the user selected option. If the option value is greater than the message level, the message is displayed. For example, the following message would always be displayed:

```
ose_out_debug(message,-1);
```

A message with a level of 2 would be displayed only if the All Debug Statements option was selected.

Milestone Message Routine

The `ose_out_milestone()` function is used to send information that may be of general interest to the user. This function has the following format:

```
ose_out_milestone(message,level);
char *message;
int level;
```

The first argument, `message`, is a pointer to a string containing the milestone message to be displayed. Newline characters are not required at the end of the string because the Online Exerciser control program formats the string into a statement containing the test program ID, time, and other useful information.

The second argument, `level`, is an integer from -1 to 2. This value represents the relative importance of the message. Four options for the `Milestone Statements` parameter are available from the `Param` menu. Each option is assigned a value as follows:

Value	Option
0	`Only Mandatory Milestones`
1	`Mandatory and Major Milestones`
2	`Most Milestones`
3	`All Milestones`

When a message is passed to the Online Exerciser, the control program compares the message `level` with the value of the user selected option. If the option value is greater than the message `level`, the message is displayed. For example, the following message would always be displayed:

```
ose_out_milestone(message,-1);
```

A message with a `level` of 2 would be displayed only if the `All Milestones` option was selected.

Error Message Routine

The `ose_out_error()` function is used to display a message indicating that an error occurred while a test program was run under the control of the Online Exerciser program. This function has the following format:

```
ose_out_error(message);
char *message;
```

The argument, `message`, is a pointer to a string containing the error message to be displayed. Newline characters are not required at the end of the string because the Online Exerciser control program formats the string into a statement containing the test program ID, time, and other useful information.

Abort Message Routine

The `ose_out_abort_notice()` function is used to display a message indicating that the test program is going to terminate due to an abnormal condition. This function has the following format:

```
ose_out_abort_notice(message);
char *message;
```

The argument, `message`, is a pointer to a string containing the message to be displayed. Newline characters are not required at the end of the string because the Online Exerciser control program formats the string into a statement containing the test program ID, time, and other useful information.

Configuration Variable Passing Routines

Two routines in `liboe.a` are used to pass configuration variables between the control program and test program when the configuration function of the test program is activated. The `ose_out_string_param()` function is used to pass string parameter information and the `ose_out_disc_param()` function is used to pass discrete parameter information.

Passing String Parameters

The `ose_out_string_param()` function is used to send string parameter information from the test program to the control program. This function has the following format:

```
ose_out_string_param(code,codename,length,dfltstr);
int code;
char *codename;
int length;
char *dfltstr;
```

The first argument, `code`, contains the ID of the parameter. Online Exerciser test programs can use parameter IDs greater than or equal to the constant `TEST_UNIQUE`, which is defined in `oe.h`.

Note – The string and discrete parameters must have unique IDs.

The second argument, `codename`, is a character pointer to text describing the parameter. This string should be short enough to fit in the parameter name window of the `Param` menu. Do not exceed 34 characters (not including the string termination character).

The third argument, `length`, contains the maximum number of characters that this string parameter may contain.

The fourth argument, `dfltstr`, is a character pointer to text with the default value of this string parameter. This string should be short enough to fit in the parameter value window of the `Param` menu. The default value received must not exceed 34 characters (not including the string termination character). The maximum string length is 100 characters (including the string termination character).

When you are using the Param menu, only the codename and dfltstr values are displayed.

Passing Discrete Parameters

The ose_out_disc_param() function is used to send discrete parameter information from the test program to the control program. This function has the following format:

```
ose_out_disc_param(code,codename,selection,selectname);
int code;
char *codename;
int selection;
char *selectname;
```

The first argument, code, contains the ID of the parameter. Online Exerciser test programs can use parameter IDs greater than or equal to the constant TEST_UNIQUE, which is defined in oe.h.

Note – The string and discrete parameters must have unique IDs.

The second argument, codename, is a character pointer to text describing the parameter. This string should be short enough to fit in the parameter name window of the Param menu. Do not exceed 34 characters (not including the string termination character).

The third argument, selection, contains the ID of a value to which this parameter can be set. The ose_out_disc_param() function must be called with each possible selection of this parameter.

The fourth argument, selectname, is a character pointer to the string containing the selection value. This string should be short enough to fit in the parameter value window of the Param menu. The default value received must not exceed 34 characters (not including the string termination character). The maximum string length is 100 characters (including the string termination character).

When you are using the Param menu, only the codename and selectname values are displayed.

Execute_Test() Routine

The Execute_Test() function in liboe.a allows you to easily port test code to run under the control of the Online Exerciser program. This function is called within the test function of the test program and is used to execute code external to the test function. The external test code should be written to send error messages to stderr. Milestone messages are any messages sent to stdout. This function has the following format:

```
Execute_Test(Cmd);
char *Cmd;
```

The argument, Cmd, is a pointer to the string containing the name of the executable and its command line arguments.

main() Routine

The main() routine is made a part of the test program when the test code is compiled with liboe.a.

CHAPTER 4

Sample Test Program

An Online Exerciser test program contains a minimum of five functions, four of which may be stubs, and two structures. The required functions and structures are described in Chapter 2. The `main()` function of the test program is provided when the code is compiled with `liboe.a`.

The `liboe.a` file also provides library routines that can be used in a test program to report messages, pass parameters, and execute code outside a test function. These routines are described in Chapter 3.

A sample bare minimum test program, `template.c`, is provided in the `/usr/oe/test/devel` directory. This program contains the required functions and structures and can be used by a test developer to become familiar with building and executing a test program under the control of the Online Exerciser.

Generating an Executable Image

Before you can incorporate a new test program into the Online Exerciser file structure, you must make the program executable. For example, to generate an executable image of the sample program, `template.c`, type:

```
cc -o template template.c -I/usr/include -L/usr/lib -loe
```

This creates an executable image called `template` using default pathnames. You can also use the makefile in the `/usr/oe/test/devel` directory to perform the same operations.

Note – Check the permissions for `template` and verify that it is an executable file before continuing.

Adding a Test Program to the File Structure

Once you have an executable image, it must be added to the Online Exerciser file structure and identified in the test library. The following steps were performed by the test code developer to incorporate the `template` test program. You can perform similar steps to incorporate additional test programs.

1. Created a directory called `devel` in `/usr/oe/test`.
2. Installed the executable image, `template`, in `/usr/oe/test/devel`.
3. Moved to `/usr/oe` and edited the file `testlib`.
4. Added `template` and its test directory to the test library file, `testlib`. For example, the new test program could have been added after the entry for the memory test as follows:

```
memtest    memory
template   devel
```

Executing the Test Program

Once the new test program is added to the test library, perform the following steps to execute the program under the control of the Online Exerciser:

1. Type **`./oe`** in the `/usr/oe` directory to bring up the Online Exerciser.
2. Select the `Group` option.

3. Select the Add option. The sample test program should be listed as a test option that can be run under the Online Exerciser program. For example, the following will be displayed:

```
hostname Exerciser Template devel template
```

where *hostname* specifies the name of the system containing the test.

4. Create a test group containing the new program.

5. Select the Execute option to run the test group.

Once the test program starts, milestone messages are displayed in the following format:

```
S hostname template id date time Exerciser Template() Started
M hostname template id date time P1 C1 starting test
M hostname template id date time P1 C1 start of pass initialization started
M hostname template id date time P1 C1 start of pass initialization completed
M hostname template id date time P1 C1 Test Function start
M hostname template id date time P1 C1 this is a bare minimum OE test
C hostname template id date time P1 C1 Case 'Test Function' Complete
M hostname template id date time P1 C2 end of pass cleanup started
M hostname template id date time P1 C2 end of pass cleanup completed
P hostname template id date time P1 Pass Complete
F hostname template id date time Exerciser Template() Completed
```

Variable	Description
hostname	Specifies the name of the system where the test is running.
id	Specifies the ID number assigned to the test.
date	Specifies the date the message was received.
time	Specifies the time the message was received.

template.c Test Program

CODE EXAMPLE 4-1 displays the contents of the sample test program, template.c, which contains the required functions and uses the required structures.

CODE EXAMPLE 4-1 Sample Test Program

```
#include <stdio.h>
#include <oe.h>

struct t_result *test_config()     /* Configuration Function */
{
 static struct t_result result;
 result.r_code = PASSED;
 result.r_message = NULL;
 return(&result);
}

struct t_result *test_param(type,selection,strptr) /* Set
Parameters */
/* Function */
int type;
int selection;
char *strptr;
{
 static struct t_result result;
 result.r_code = PASSED;
 result.r_message = NULL;
 return(&result);
}

struct t_result *test_init()     /* Initialization Function */
{
 static struct t_result result;
 result.r_code = PASSED;
 result.r_message = NULL;
 retunr(&result);
}

struct t_result *test_cleanup()     /* Cleanup Function */
{
 static struct t_result result;
 result.r_code = PASSED;
 result.r_message = NULL;
 return(&result);
}
struct t_result *Test()              /* Test Function */
{
 static struct t_result result;

 ose_out_milestone("this is a bare minimum OE test",0);

 result.r_code = PASSED;
 result.r_message = NULL;
 return(&result);
}

struct ose_test test[] = {          /* Test Array */
 NULL,"Exerciser Template",NULL,
 *test_config,"Test Configuration",NULL,
 *test_param,"Test Parameters",NULL,
 *test_init,"Test Initialization",NULL,
 *test_cleanup,"Test Cleanup",NULL,
 *Test,"Test Function",TEST_ENABLE,
 NULL,NULL,NULL
};
```