LabWindows®/CVI
Test Executive Toolkit
Reference Manual

November 1994 Edition

Part Number 320863A-01

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20, Canada (Ontario) (519) 622-9310,
Canada (Québec) (514) 694-8521, Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921, Mexico 95 800 010 0793,
Netherlands 03480-33466, Norway 32-84 84 00, Singapore 2265886, Spain (91) 640 0085, Sweden 08-730 49 70,
Switzerland 056/20 51 51, Taiwan 02 377 1200, U.K. 0635 523545

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

Figures

Tables

About This Manual

The *LabWindows/CVI Test Executive Toolkit Reference Manual* describes the LabWindows/CVI add-on package you can use for automated sequencing of test programs.

Organization of This Manual

This manual is organized as follows:

Chapter 1, *Introduction*, describes the installation procedure and lists the main features of the Test Executive Toolkit. It also explains the execution model of the toolkit and the three operating levels.

Chapter 2, *Getting Started*, introduces the basic concepts of test executive operation and test sequence development.

Chapter 3, *Operating the Test Executive*, shows how to use the Test Executive main panel—the controls, indicators, and operator dialog boxes. The main panel is the user interface for both development and run-time operation.

Chapter 4, *Creating Tests and Test Sequences*, describes the process of creating new functions and sequences for execution by the Test Executive.

Chapter 5, *Modifying the Test Executive,* describes the internal structure of and how to modify the Test Executive.

Chapter 6, *Function Descriptions for the Test Executive*, describes the functions in the Test Executive engine.

Appendix A, *Error Codes and Engine Constants*, lists the error codes and other important constants used by the Test Executive engine.

Appendix B, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.

The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

bold	Bold text denotes parameters, or the introduction of menu or function panel items.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, variables, filenames, and extensions, and for statements and comments taken from program code.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate word or value in the place of these terms–for example, *`filename`*.
<Ctrl>	Key names in the text match the names on the keys. For example, the Control key appears in the text as <Ctrl>.
-	A hyphen between two or more key names denotes that you should simultaneously press the named keys–for example, <Ctrl-Alt-Del>.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- Your *Getting Started with LabWindows/CVI* manual
- Your *LabWindows/CVI User Manual*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and technical support forms for you to complete. These forms are in the Appendix B, *Customer Communication*, at the end of this manual.

Chapter 1
Introduction

This chapter describes the installation procedure and lists the main features of the Test Executive Toolkit. It also explains the execution model of the toolkit and the three operating levels.

Installation

The following section contains instructions for installing the Test Executive on the Windows and Sun SPARCstation platforms.

The LabWindows/CVI Test Executive comes in compressed form on a floppy disk. Installing the Test Executive requires approximately 1.2 MB.

Some virus detection programs may interfere with the installer program. Check the distribution disk for viruses before you begin installation. Then turn off the automatic virus checker and run the installer. After installation, it is good practice to check your hard disk for viruses again and then turn on the virus checker.

Windows

You can install the Test Executive from the Windows File Manager or with the **Run...** command from the **File** menu of the Program Manager.

Insert the Test Executive Toolkit disk into the 3.5-inch disk drive and run the `SETUP.EXE` program using one of the following three methods.

- Under Windows, launch the File Manager. Click on the drive icon that contains the installation disk. Find `SETUP.EXE` in the list of files on that disk and double-click on it.
- Under Windows, select **Run...** from the **File** menu of the Program Manager. A dialog box appears. Type `X:\SETUP` (where `X` is the proper drive designation).

After you choose an installation option, follow the instructions that appear on the screen.

SPARCstation

Perform the following steps to install the Test Executive. You do not need root privileges to install the Test Executive, but you must be able to write to the LabWindows/CVI directory where the Test Executive will be installed.

1. Connect to the LabWindows/CVI directory.

2. Insert the Test Executive disk into the 3.5-inch disk drive.

3. Type the following UNIX command: `tar xvf /dev/fd0`

After you install the LabWindows/CVI Test Executive, your LabWindows/CVI directory should contain a new `testexec` directory.

Updating Sequence Paths

If you installed the Test Executive on a SPARCstation or in a non-default directory under MS-Windows, you must perform the following steps to update the paths in the sequence files so that you will be able to run the example sequences.

1. Start LabWindows/CVI and open and run the project `testexec.prj`. When the Login dialog box appears, type the word `developer` in the password box and click on **OK**. The main panel of the Test Executive appears.

2. Choose **Open...** from the **File** menu and select the file `computer.squ` in the Load Test Sequence dialog box.

Note: ***You can ignore any error messages which begin*** `Unable to verify test`….

3. Select **Edit Sequence...** from the **Sequence** menu. The Sequence Editor appears.

4. In the Sequence Editor, click on the **Edit Paths...** button. The Edit Paths dialog box appears.

5. Perform the following steps for each item in the Current Paths in Sequence list box.

   a. Select the item and click on **Update Selected File…**. The Select Replacement dialog box appears.

   b. In the dialog box that appears, select the file name that matches the file you selected in the Current Paths in Sequence list box and click on **OK.**

6. Click on **OK** in the Edit Paths dialog box and then click on **OK** in the Sequence Editor.

7. Select **Save** from the **File** menu in the Test Executive main panel.

8. Choose **Open…** from the **File** menu. Update the sequence paths for each of the example sequence files as described in the preceding steps.

Product Overview

The LabWindows/CVI Test Executive Toolkit creates an automated test system. This add-on package includes a complete Test Executive that can perform many standard operations. The toolkit includes source code, so you can change or expand the functionality.

The Test Executive runs test programs using C functions. The following are the main features of the Test Executive.

- Test sequencing based on pass/fail status, preconditions, and goto commands
- Logging of test results
- Generating ASCII test reports to file
- Run-time interfacing, including prompts for operator and (UUT) Unit Under Test serial number, Pass and Fail banners, and run-time error notification
- Forcing individual tests to pass, fail, or skip for test sequence debugging
- Halting and looping on individual test failure
- Testing continuously in UUT mode
- Running pre- and post-run routines for system setup and shutdown
- Using three operating levels

Execution Model

The Test Executive can execute a sequence in one of three ways—UUT Test, Single Pass, or Single Test. The UUT Test, invoked when the operator clicks on the **Test UUT** button, executes a test sequence repetitively. Each test cycle includes a UUT serial number prompt and Pass, Fail, and Abort banners to notify the operator of the test result for the current UUT.

UUT Test mode is the production operating mode for testing multiple UUTs. You use Single Pass mode primarily during development and sometimes for diagnostic purposes. In Single Pass mode, the test sequence executes only once, and you are not prompted for the UUT serial number or with the Pass, Fail, and Abort banner display. The following illustration shows the overall flow of execution in UUT Test and Single Pass operation.

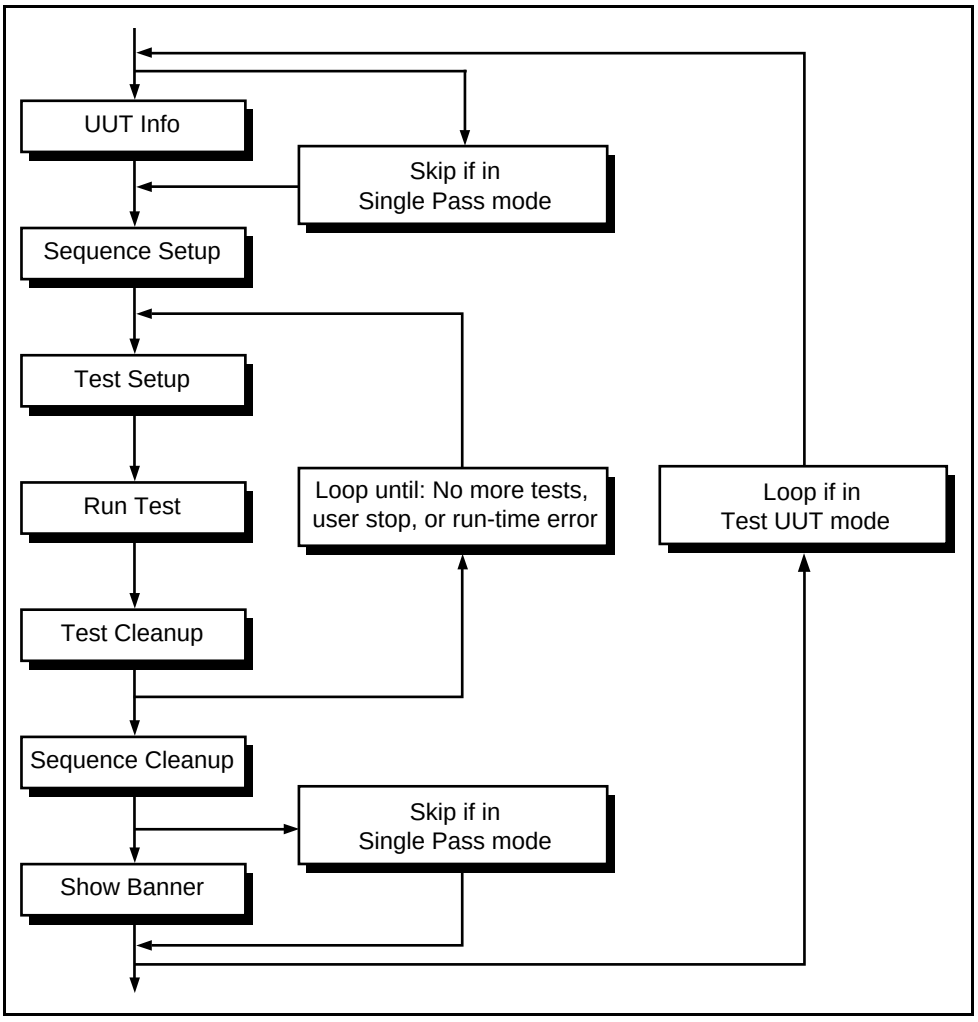

Figure 1-1. Flowchart for UUT Test and Single Pass Operation

In addition to UUT Test and Single Pass operation, you can choose Single Test execution to run an individual test. You should use Single Test execution primarily for diagnostic purposes.

Operating Levels

The Test Executive has three operating levels—Developer, Technician, and Operator. The following table summarizes the capabilities available in each operating level.

Table 1-1. Text Executive Operating Levels

Level	UUT Test	Single Pass/Single Test	Edit Sequences
Developer	Yes	Yes	Yes
Technician	Yes	Yes	No
Operator	Yes	No	No

At the Developer level, you have access to all capabilities of the Test Executive.

At the Technician level, you can run individual tests, but you cannot edit sequences. You can also run a sequence in Single Pass mode. The Technician level gives you the flexibility to execute tests to diagnose a UUT.

The Operator level is the most restrictive, in that you can execute only test sequences in UUT Test mode by selecting the **Test UUT** button.

When you load the Test Executive, set the operating level by typing one of three passwords in the Login dialog box: `developer`, `technician`, or `operator`. While the Test Executive runs, you can change the operating level by re-opening the Login dialog box under the **File** menu.

Development Model

The LabWindows/CVI Test Executive is a completely functional, ready-to-run test executive. Chapter 3, *Operating the Test Executive*, describes the operation of the test executive at the highest level.

The Test Executive is contained in the LabWindows/CVI project file `testexec.prj`. You may want to make copies of `testexec.prj` with different names for each of your development efforts.

`testexec.prj` is a project file that contains all the source code files for the Test Executive. You can build a standalone executable from the `testexec.prj` file. You can include your test functions as source code files in the `testexec.prj` file and when you create a standalone executable. However, programmers normally maintain their tests as separate object or library files, keeping them separate from the Test Executive. Because LabWindows/CVI executes your tests using the Utility Library functions `LoadExternalModule` and

`GetExternalModuleAddr`, you can develop and distribute the object or library files for your tests separately from the standalone executable.

You can make the `testexec.prj` project into a standalone executable without modification. Later, you develop tests independently, using LabWindows/CVI, and save them as object or library files. The Test Executive can load these tests dynamically, leaving the executable unchanged.

Perform the following steps to develop your test functions for the LabWindows/CVI Test Executive.

1. Develop each new test as a C function in source code. In order to integrate your test properly with the Test Executive, your test functions must follow the prototypes shown in the LabWindows/CVI example tests. (The file `template.c` contains standard prototypes.)

2. Temporarily add your source code files to the project (`testexec.prj`) so that you can test them using the source code debugging features of LabWindows/CVI.

3. Compile the source code into an object module using LabWindows/CVI or the Watcom, Borland, or Symantec 32-bit C compilers.

4. Remove the source code files from the project file.

You use the Sequence Editor of the Test Executive to create a test sequence that contains your tests. In the Sequence Editor, you can configure the run options and test preconditions that control the flow of your test sequence. You can also specify the name of the test function and the file in which it exists.

Chapter 4, *Creating Tests and Test Sequences,* tells you how to create new test functions and test sequences. When you want to customize the LabWindows/CVI Test Executive, you can modify the Test Executive project file in LabWindows/CVI. Refer to the project file `testexec.prj` which lists all of the `.uir` files and object modules for the Test Executive. The project file also shows you the source code files that correspond to the object modules, but the system ignores these source code files when you build an executable. If you want to modify any of the source code files for the Test Executive, include the appropriate source code file in the project list and exclude the corresponding object files.

Chapter 5, *Modifying the Test Executive*, covers the following topics:

- Organization of source code files
- Common modifications
- Advanced modifications
- Writing a test executive

If you want to write your own test executive, the instrument driver library—specifically, the LabWindows/CVI Test Executive Engine in the files `txengine.*`—gives you a starting point. Chapter 6, *Function Descriptions for the Test Executive Engine*, describes the functions you can use to build your test executive engine.

Chapter 2
Getting Started

This chapter introduces the basic concepts of test executive operation and test sequence development, and contains the following examples.

- Operating a Test Sequence
- Examining a Test Function
- Editing a Test Sequence

You should go through these examples in the order they are presented. The first example, *Running a Test Sequence*, is relevant to anyone who operates the Test Executive. The second two examples, *Examining a Test Function* and *Editing a Test Sequence*, are for users who write test functions and create test sequences.

The Test Executive comes with four test sequence examples, which are described at the end of this chapter.

This chapter assumes that you are running the Test Executive within the LabWindows/CVI environment. If you are executing the Test Executive as an standalone application, you can only use the first example–*Running a Test Sequence*.

Operating a Test Sequence

Starting the Test Executive

Perform the following steps to start the Test Executive.

1. Launch LabWindows/CVI.
2. Open the Test Executive project, which can be found in `testexec.prj` to start the Test Executive. The setup program places this file in the `TESTEXEC` directory.
3. Run the Test Executive by choosing **Run Project** from the **Run** menu. The Login dialog box appears.

4. Type in your name and password as shown in the following illustration.

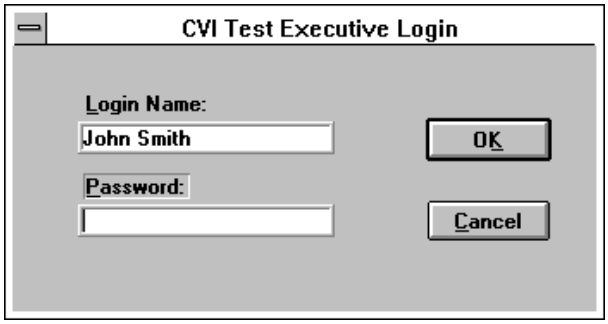

Figure 2-1. Login Dialog Box

The Password field sets the operating level of the Test Executive. For this session, you will start the Test Executive at the Operator level, so type `operator` in the Name and Password fields.

Note: ***You can access the Operator level by typing any text in the Password field. However, you must type*** `technician` ***to access the Technician level and*** `developer` ***to access the Developer level.***

5. Click on **OK** to confirm your entries.

On the Test Executive front panel, notice the word Stopped that appears next to the large LED. This LED is the Status indicator. Stopped indicates that no test sequence is currently running. Your screen appears as shown in the following illustration.

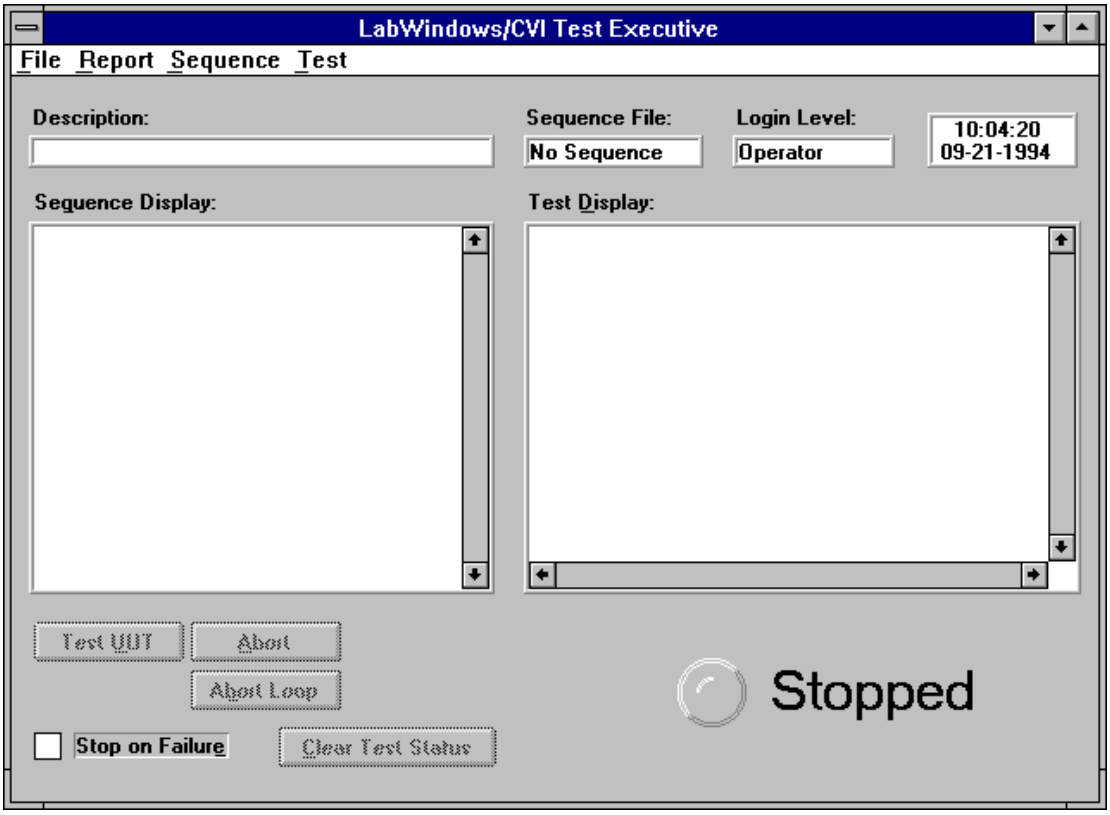

Figure 2-2. Test Executive Front Panel in Stopped Mode

Running a Test Sequence

Perform the following steps to run a test sequence.

1. Select **Open...** from the **File** menu.

2. Select the file `COMPUTER.SQU`, which can be found in the `EXAMPLES` subdirectory of the `TESTEXEC` directory.

   When you load `COMPUTER.SQU`, notice that the test sequence appears in the Sequence Display to execute the sequence. Also, the name of the test sequence appears at the top of the Test Executive main panel.

3. Click on the **Test UUT** button located beneath the Sequence Display to execute the sequence. The Test Executive prompts you for the serial number of the UUT.

4. Type `123` in the input box and click on the **OK** button. The simulator asks you to specify which tests, if any, will fail. Select a few of the tests and then click on **Done**. The sequence executes.

   As the sequence executes, notice the following events on your screen.

   - The Status indicator displays `RUNNING` with a flashing LED.
   - As each test runs, the word `RUNNING` appears next to the name of the active test in the Sequence Display.
   - After each test runs, the Pass, Fail, or Skip status of the test appears next to the name of the test in the Sequence Display, and the test result appears in the Test Display.

   When the sequence execution completes, a banner appears indicating whether the UUT passed, failed, or was aborted.

5. Click on **OK** in the completion banner, and enter another serial number when the UUT Information dialog box appears. The Test Executive continues to cycle to the next UUT until you click on **Stop** in the UUT Information dialog box.

6. To view the test report after execution completes, choose **View** from the **Report** menu. The report appears in the Test Display string indicator.

The Test Report includes the name and description of the sequence, the date and time that testing began, the operator's name, and the results of testing for each UUT. When you set this test sequence to generate a test report file, the system automatically writes the test report to the file each time the sequence executes.

Changing to Technician Level

Perform the following steps to change from Operator to Technician level and to see the more flexible execution capabilities at the Technician level.

1. Select **Login…** from the **File** menu.

2. In the Login dialog box, type the word `technician` in the Password field and click on **OK.** The **Single Pass, Run Test**, and **Loop Test** buttons appear in the lower left corner of the Test Executive front panel.

3. If you do not see these buttons, select **Login…** from the **File** menu again and carefully type the word `technician` in the Password field again.

How to Run Single Tests and Use Single Pass Mode

To run a single test, click on the name of the desired test and then the **Run Test** button. Notice that only the selected test runs. Single Test execution selectively runs individual tests for diagnosis and troubleshooting. The test status appears next to the test name in the Sequence Display. You can also execute a test repeatedly by clicking on the **Loop Test** button.

Now click on the **Single Pass** button. Clicking on this button runs the entire test sequence one time without the UUT prompts and banners. Notice that the sequence stops after it executes one time. When you select **View** from the **Report** menu, the Test Executive generates the Test Report as it did at the Technician level.

Exiting The Test Executive

Select **Exit** from the **File** menu to stop execution of the Test Executive.

Examining a Test Function

The following example shows you a simple function that you can execute in a test sequence. You need to study this example only if you write test functions and incorporate them into the Test Executive. You need LabWindows/CVI programming experience to complete this example. More information on test functions and sequencing appears in Chapter 4, *Creating Tests and Test Sequences*.

Perform the following steps to learn about how a test function is built.

1. Launch LabWindows/CVI if you have not already done so, but do not run the Test Executive yet.

2. Find and open the source file `random.c`, located in the `TESTEXEC/EXAMPLES` directory.

The `Random` function in this file illustrates the basic structure of a functioning test function in the Test Executive. `Random` needs two parameters, data and error, as shown in the following code segment.

```
void Random(tTestData *data, tTestError *error)
{
   double measurement, limit;
   srand(clock());
   limit = (double)rand() / RAND_MAX;
   measurement = (double)rand() / RAND_MAX;
   error->errorFlag=FALSE;
   if (limit >= measurement)
      data->result = PASS;
   else
      data->result = FAIL;
   data->measurement = measurement;
   data->outBuffer = malloc(100);
   sprintf(data->outBuffer,"measurement %f, limit %f",
      measurement, limit);
}
```

The `tTestData` structure transmits information about the result of the test. The `tTestError` structure transmits information the Test Executive uses for run-time error handling. In this

example, you use `Random` as if it were a newly created test function to step through the test sequence creation process.

`Random` generates two random numbers, limit and measurement. The function compares measurement to limit, setting the `result` member in the `tTestData` structure to the result of the comparison. This function also passes one of the random numbers and a comment as the `measurement` and `outBuffer` members of the `tTestData` structure. When you create a test sequence that calls this function later in this chapter, you will see how these structure members are used.

3. Close `random.c`. Do not save any changes.

You can now use this test function to create a test sequence.

Editing a Test Sequence

The following steps show you how to set up and edit a test sequence.

1. Launch LabWindows/CVI if you have not already done so.

2. Run the Test Executive. Type `developer` in the Password field of the Login dialog box and click on **OK**.

3. Select **Edit Sequence…** from the **Sequence** menu to invoke the Sequence Editor.

Notice that the list box at the top of the Sequence Editor dialog box is empty. This list box shows the tests that are defined for the current sequence. Use the Sequence Editor to input all of the specifications required to define a test sequence.

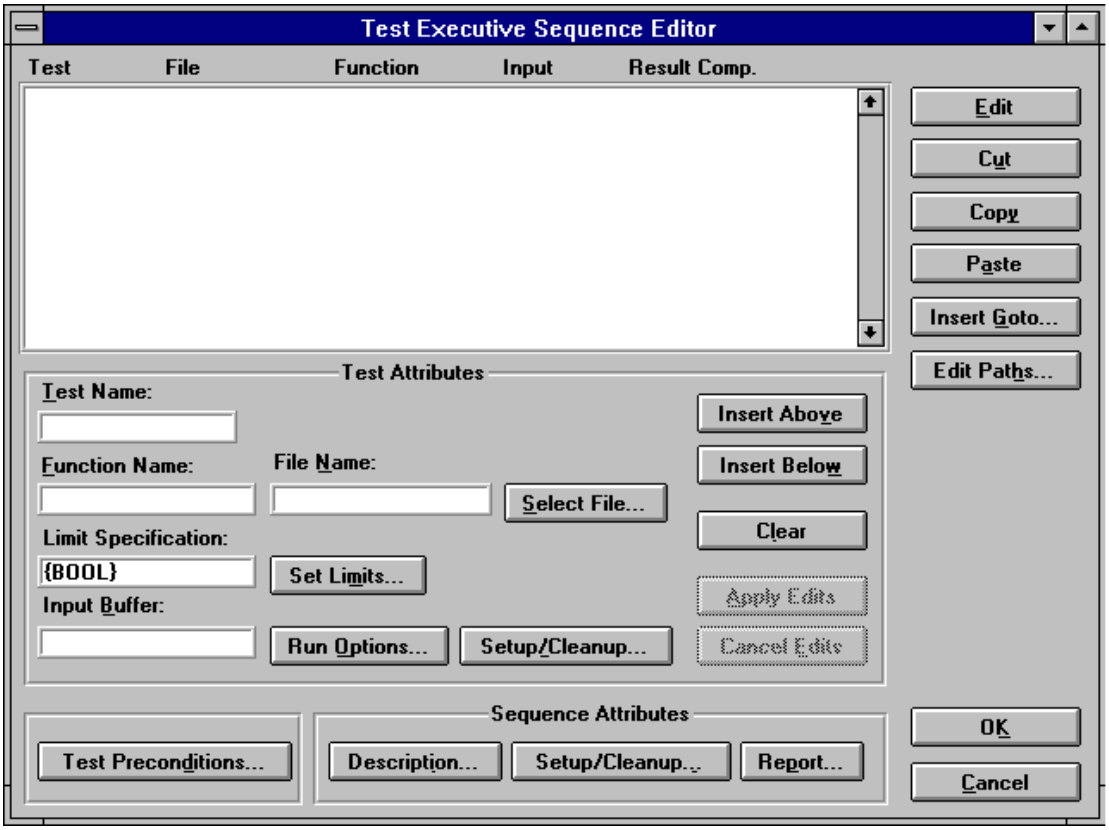

Figure 2-3. Sequence Editor Dialog Box

4. To add tests to a sequence, you enter parameters in the Test Attributes controls located below the Sequence Editor list box. Click in the control labeled Test Name. In this example, you add the `Random` function to a new sequence and configure its limit specifications. Type `Random-Boolean` in the Test Name control.

5. Click on the **Select File** button to choose the file containing the function you want to run for this test. For this example, select `random.obj`. Now type `Random` in the Function Name control.

   To determine if a test passes or fails, the Test Executive must have a limit specification. The Test Executive looks at the `tTestData` structure of the test function and applies the limit checking you specify in the sequence editor to each test. The `tTestData` structure contains both a Boolean flag and a numeric measurement. You can use either of these elements to determine if the function passes.

6. Click on the **Set Limit…** button to view the Set Limit Specification dialog box. Scroll through the Comparison Type ring control to see the available types of checking. If you choose a numeric comparison, you must also enter the numeric limits used for the

comparison. For this example, set Comparison Type to `BOOL`. Your Set Limit Specification dialog box should look like the following illustration.

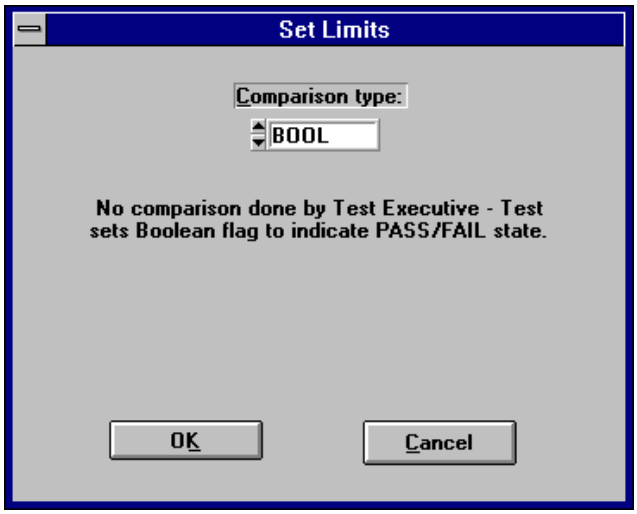

Figure 2-4. Set Limits Specification Dialog Box

Setting Comparison Type to `BOOL` sets the Test Executive to use the result flag in the `tTestData` structure to determine if the test passed or failed.

7. Click on **OK** to confirm the limit specification. The Limit Specification control should now contain the text `{BOOL}`.

8. Click on **Insert Below** to add the test to the sequence. The new test information appears in the Sequence Editor list box.

9. Next, add another test that uses a numeric limit specification, rather than a Boolean. Notice that all the fields remain filled in. Because you are using the function `Random` again, you need to make only a few changes. Change the Test Name to `Random-Numeric`.

10. Click on **Set Limit** and set the Limit Specification to the numeric comparison, `GELE`, which means greater than or equal to a lower limit and less than or equal to an upper limit. Set the lower limit to `0.00` and the upper limit to `0.50`, as shown in the following illustration.

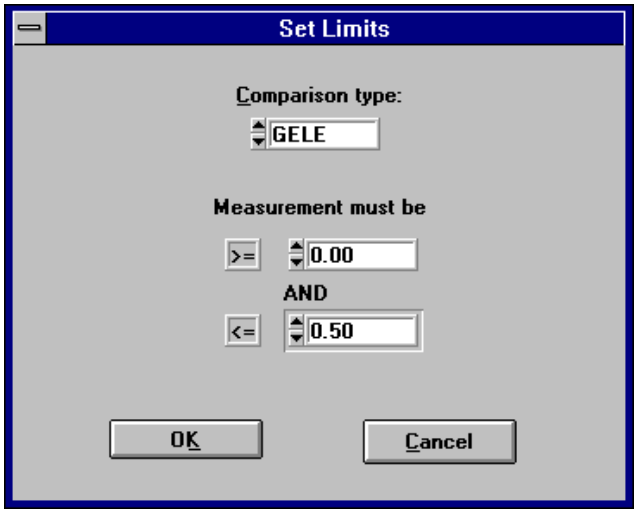

Figure 2-5. Setting Number Limits

11. Click on **OK** to accept the limit specification.

12. Click on **Insert Below** to add this test after the `Random-Boolean` test. Your completed test sequence should match the Sequence Editor list box shown in the following illustration.

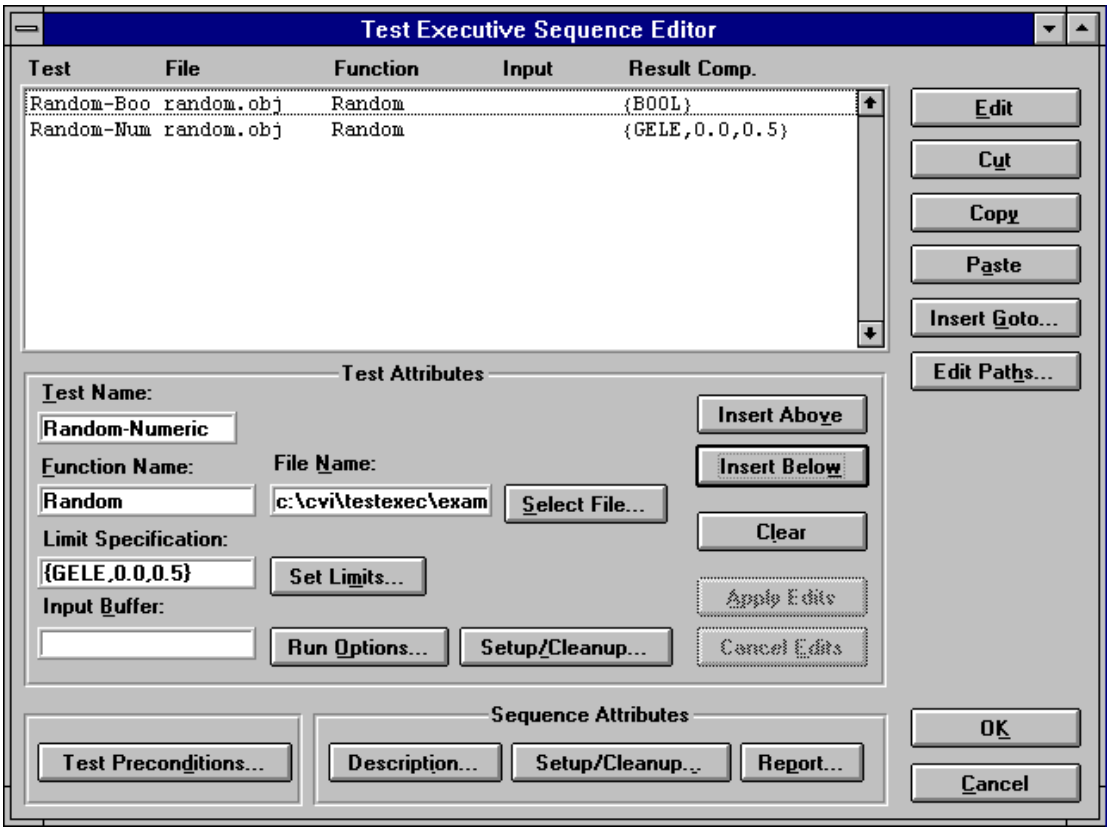

Figure 2-6. A Completed Test Sequence Setup

13. To modify the definition of a test, click on the test you want to modify in the Sequence Editor list box. Then click on **Edit**. The specifications of the test appear in the Test Attributes controls (the Test Name, Function Name, Input Buffer, Limit Specification, Run Options and Setup/Cleanup dialog boxes). Make any changes and click **Apply Edits** to overwrite the existing information.

Setting Preconditions

In the following steps you set up a dependency between the two tests in your sequence.

1. Click on the **Test Preconditions** button. The Precondition Editor window appears.

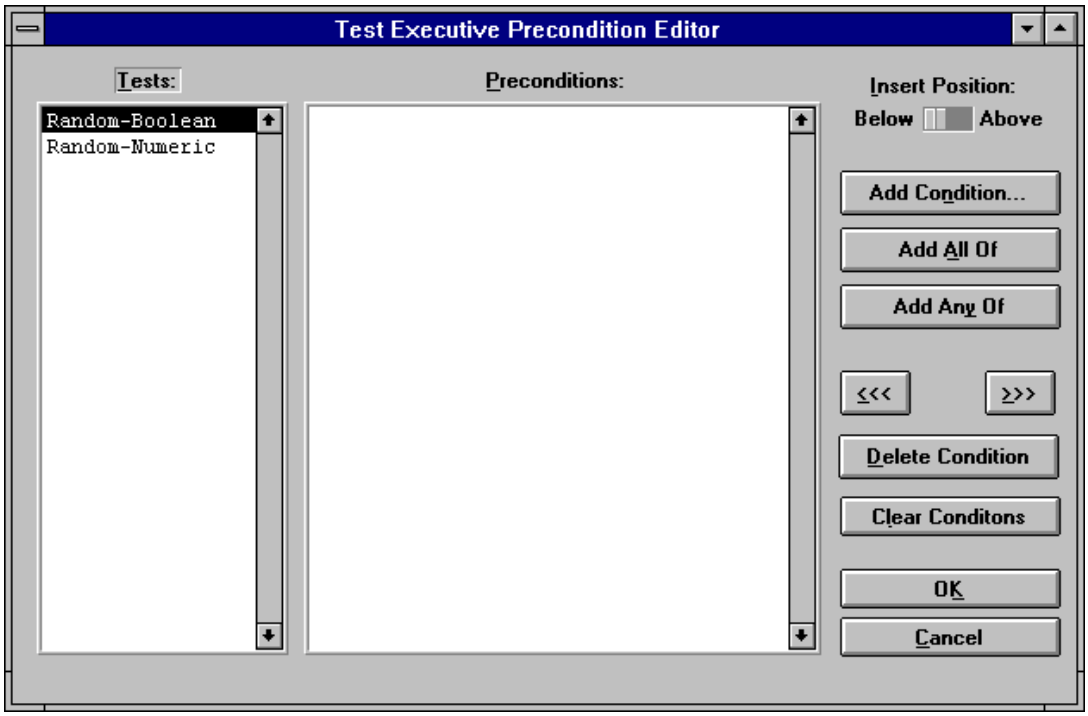

Figure 2-7. Precondition Editor

2. Specify that `Random-Boolean` must pass for `Random-Numeric` to execute. Click on `Random-Numeric` in the Tests list box. Click on **Add Condition...**. The Add Condition dialog box appears.

3. In the Add Condition dialog box, set the Type switch to `PASS` and click on `Random-Boolean` in the Tests list box. A check mark appears beside the name. Click on **OK**.

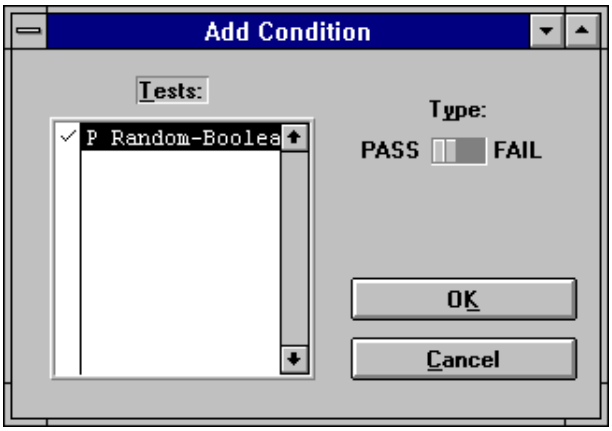

Figure 2-8. Using Add Condition to Set Preconditions

The Preconditions list box now shows `PASS Random-Boolean`. This signifies that `Random-Numeric` runs only if the precondition test `Random-Boolean` passes.

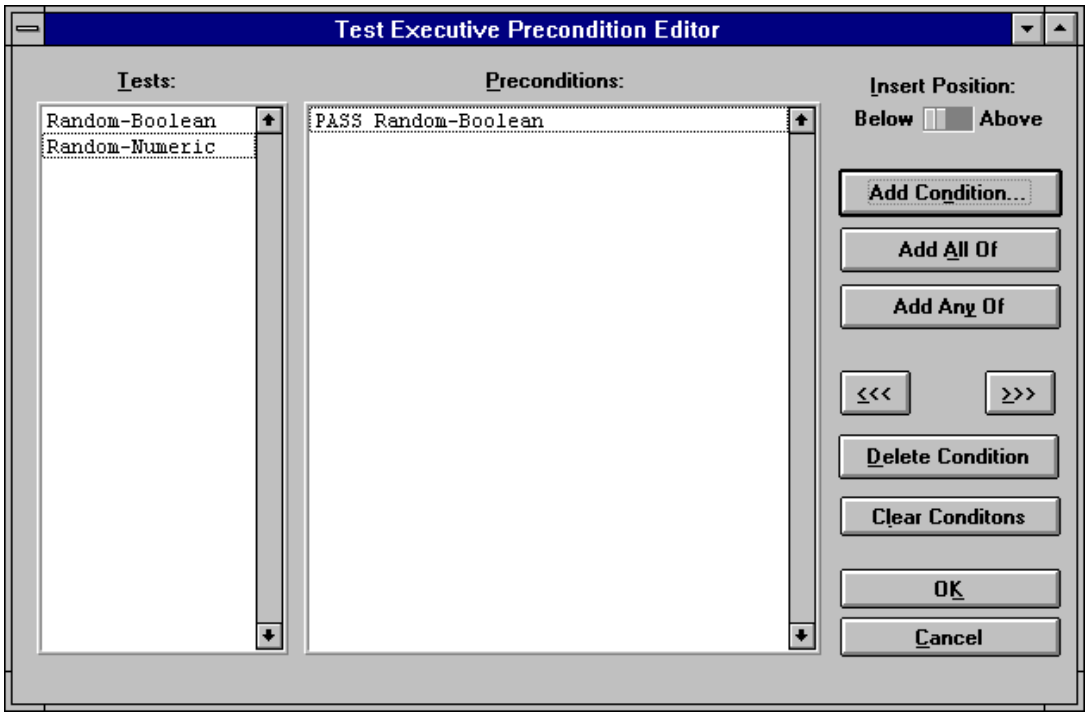

Figure 2-9. A Completed Random-Boolean Precondition Setup

4. Click on **OK** to save the new dependency specification and return to the Sequence Editor.

Running the Sequence

You are now ready to run your test sequence.

1. Return to the main Test Executive front panel by clicking on **OK**. The new test sequence appears in the Sequence Display.

2. Click on the **Test UUT** button to run the sequence. The Test Executive automatically determines if the test passes or fails based on the values transmitted in the `tTestData` structure. Perform the following steps to see your specification in action.

   a. After you have tested several UUTs, click on the **Stop** button in the UUT Information dialog box.

   b. Select **View** from the **Report** menu to see the data generated for each test by the `Random` function.

3. Choose **Exit** from the **File** menu to quit the Test Executive. The application automatically prompts you to save the sequence you created. Do not save your changes.

The three examples presented in this chapter show the fundamental operations in the Test Executive: developing test functions and using the Sequence Editor to create sequences that use these functions. The remaining chapters describe the Test Executive in greater detail.

Example Sequences

The Test Executive package includes four test sequence examples located in the `EXAMPLES` subdirectory of the Test Executive. The four sequences, `COMMENT.SQU`, `COMPUTER.SQU`, `RTERROR.SQU`, and `PREPOST.SQU`, demonstrate different aspects of the Test Executive.

`COMMENT.SQU` executes tests that use the **outBuffer** field to log test results in a customized format. When you run `COMMENT.SQU`, the Test Report contains multiple, custom-formatted lines rather than the standard formatted lines.

`COMPUTER.SQU` includes setup and cleanup functions (functions that run before or after a test or test sequence), multiple dependencies, and tests that use a variety of comparison types.

`RTERROR.SQU` contains the same tests as `COMPUTER.SQU`, but generates a run-time error during the test to illustrate the Run-time Error dialog boxes.

`PREPOST.SQU` demonstrates the different levels of setup and cleanup functions.

Chapter 3
Operating the Test Executive

This chapter shows how to use the Test Executive main panel—the controls, indicators, and operator dialog boxes. The main panel is the user interface for both development and run-time operation. The following illustration shows the Test Executive main panel when you are at the Developer level.

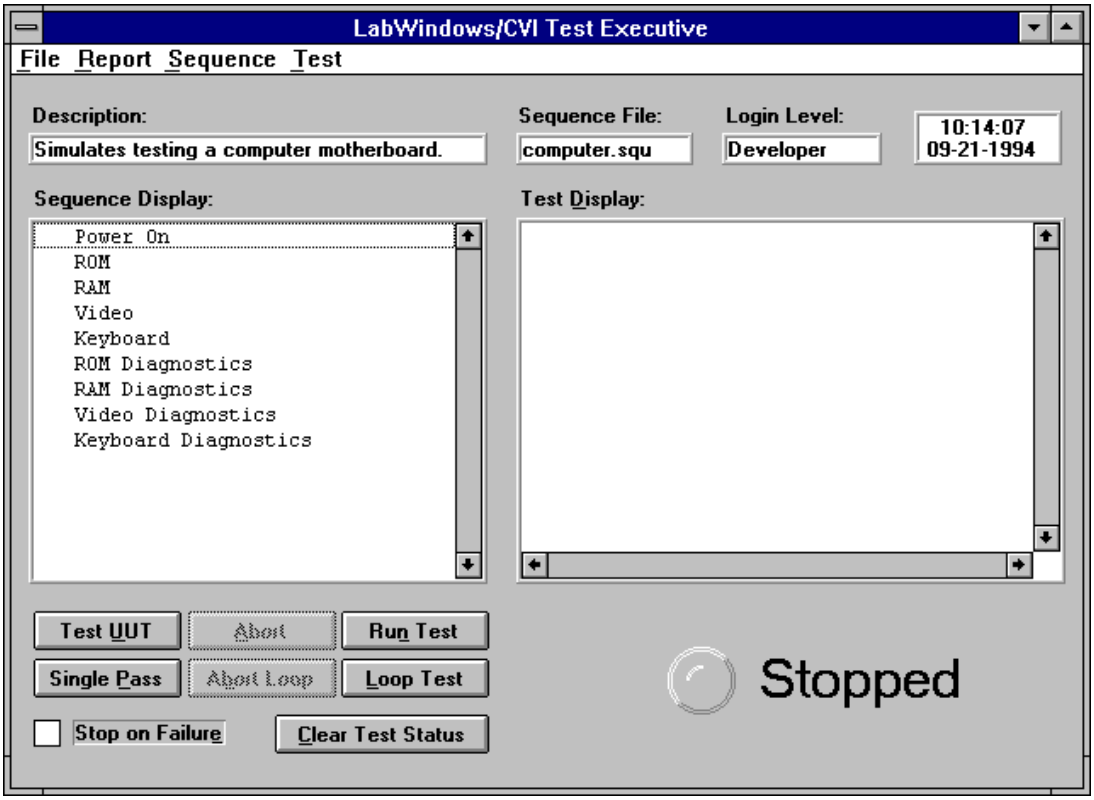

Figure 3-1. Test Executive Main Panel

Features of the Main Panel

The menu items, buttons, and other controls on the Test Executive main panel access the following three areas of operation.

- Sequence file operations and login
- Execution
- Display

The following sections describe how the Test Executive controls operate.

File Menu

Login

Choose the **Login** menu item from the **File** menu to enter a new user name and/or password. You can access the **Login** menu at any operating level.

New

The **New** menu item deletes the current sequence and clears the Test Executive Sequence Display so that you can create a new test sequence. If you have modified the current sequence, the Test Executive prompts you to save changes before loading the new sequence. You can use the **New** menu item only in the Developer operating level.

Open…

Select **Open…** from the **File** menu. The File dialog box should appear on your screen. Use this box to load a test sequence into memory from a file. Selecting a valid sequence file deletes the current sequence in memory and loads the new sequence into the Test Executive. If you have modified the current sequence, the Test Executive prompts you to save changes before loading the new sequence.

Save

The **Save** menu item saves the current test sequence. You can use the **Save** button only in the Developer operating level.

Save As…

Choose the **Save As…** menu item and the File Dialog box appears on your screen. This box saves a test sequence to file. The File Dialog box prompts you with the pathname of the sequence currently in memory, if that sequence has a pathname. You can use the **Save As…** button only in the Developer operating level.

Exit

The **Exit** menu item causes the Test Executive to stop execution. If you have modified the current sequence, the Test Executive prompts you to save your changes before quitting. You can access the **Exit** menu item in any operating level.

Report Menu

Clear

The **Clear** menu item removes the current test report from memory. **Clear** does not erase the report file. You can access the **Clear** menu item in any operating level.

View

The **View** menu item displays the current test report in the Test Display text box. Only the 20 most recent sequence passes display. You can choose the **View** menu item from any operating level.

Sequence Menu

Edit Sequence…

The **Edit Sequence…** menu item invokes the Sequence Editor. You can access the **Edit Sequence…** menu item only in the Developer operating level.

Test Menu

Normal

The **Normal** menu item sets the run mode of the selected test to normal. You can use the **Normal** menu item only in the Developer or Technician operating level.

Force to Pass

The **Force to Pass** menu item sets the run mode of the selected test to **Force to Pass.** You can see the **Force to Pass** menu item only in the Developer or Technician operating level.

Force to Fail

The **Force to Fail** menu item sets the run mode of the selected test to **Force to Fail**. You can access the **Force to Fail** menu item only in the Developer or Technician operating level.

Skip

The **Skip** menu item sets the run mode of the selected test to **Skip**. You can use the **Skip** menu item only in the Developer or Technician operating level.

Note: ***Changing the run mode using* Normal, Force to Pass, Force to Fail *and* Skip *is equivalent to using the Sequence Editor. If you use these menu items at the Developer level, the Test Executive prompts you to save the sequence before it is unloaded.***

Main Panel Controls

Test UUT

The **Test UUT** button starts a repetitive execution of the test sequence for UUT testing. (See the *Execution Model* section in Chapter 1, *Introduction*, for information about the Test UUT mode of execution.) You can use the **Test UUT** button from any operating level.

Single Pass

The **Single Pass** button executes the test sequence one time. (See the *Execution Model* section in Chapter 1, *Introduction*, for information about the Single Pass mode of execution.) You can access the **Single Pass** button only in the Developer and Technician operating level.

Abort

The **Abort** button stops sequence execution after the current test completes execution. If you start testing by clicking on the **Test UUT** button, clicking on **Abort** stops testing on the current UUT. The system then prompts you for the next UUT serial number. The **Abort** button is available at all operating levels, but is active only when a test is running.

Abort Loop

Clicking on the **Abort Loop** button stops the loop execution. Test sequence execution then continues with the next test. You can use the **Abort Loop** button only when the Test Executive loops on a test. You can access the **Abort Loop** button from any operating level.

Run Test

The **Run Test** button executes the test selected in the Sequence Display. You can use the **Run Test** button only in the Developer and Technician operating levels.

Loop Test

The **Loop Test** button starts a repetitive execution of the test currently selected in the Sequence Display list box. You can only access the **Loop Test** button from the Developer and Technician levels. When you select **Loop Test**, the following dialog box appears.

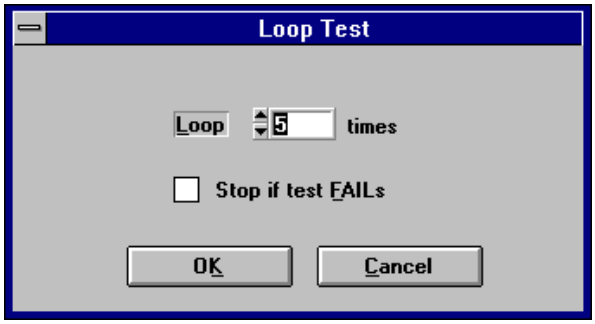

Figure 3-2. Loop Test Dialog Box

You can execute a specific number of iterations, with the option to stop if the test fails. To specify the number of iterations, enter the number of iterations in the Loop count control. The preceding figure shows a loop that will iterate five times. If you want looping to stop when the test fails, click on the Stop if test Fails check box. To confirm your inputs, click on the **OK** button. To cancel, click on **Cancel**.

Stop If Test Fails

When you select Stop if test Fails in the Loop Test dialog box, sequence execution stops when any test fails. You can access the Stop if test Fails box from any operating level.

Stop on Failure

When you select the Stop on Failure check box on the main panel of the Test Executive, the execution of test sequences stops every time a test fails.

Clear Test Status

The **Clear Test Status** button clears the Test Status/Result field for each test in the Sequence display. You can access the **Clear Test Status** button in all operating levels.

Indicators

This section describes the displays and indicators found on the Test Executive front panel.

Sequence Description

The Description indicator displays a description of the sequence, if available, above the Sequence Display.

Sequence File

The Sequence File indicator appears above the Test Display. If no sequence is loaded, or if you have not saved the current sequence, this area displays `No Sequence`. Otherwise, the Sequence File indicator displays the base filename of the test sequence.

Login Level

The Login Level indicator appears to the right of the Sequence File indicator and shows the current login level.

Sequence Display

The Sequence Display shows the list of tests in the loaded test sequence as shown in the following illustration.

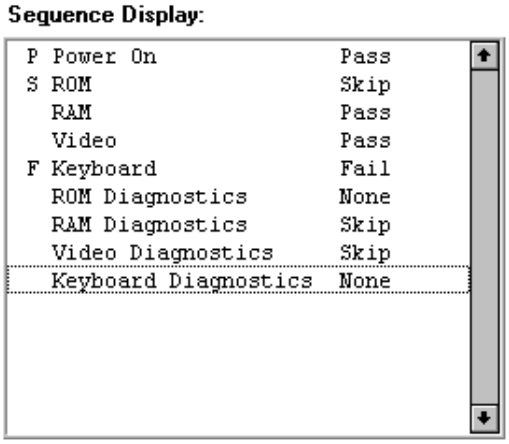

Figure 3-3. Sequence Display List Box

Each display line contains three fields. The fields in the Sequence Display are not labeled, so take special note of what field occupies each part of the line. From left to right, the fields are the run mode, Test Name, and Test Status/Result.

The run mode field indicates the setting of the run mode parameter for the test. The following table shows the available run mode values and their meanings.

Table 3-1. Run Mode Fields

Value	**Meaning**
blank (no symbol)	Test will run normally.
`S`	Test will be skipped.
`P`	Test will be skipped with a forced `PASS` result.
`F`	Test will be skipped with a forced `FAIL` result.

The Test Name field shows the name of the test.

The Test Status/Result field is set to `RUNNING` during test execution to indicate the active test. After the test completes, the field reflects the result of the test. The following table shows the possible Test Status/Result field values and their meanings.

Table 3-2. Test Status/Result Fields

Value	**Meaning**
`PASS`	Test result satisfied limit specification.
`FAIL`	Test result did not satisfy limit specification.
`SKIP`	Test did not execute.
`NONE`	Test data was logged but no comparison was made because the limit specification was set to Log only.
`ERROR`	Run-time error occurred during test execution.

Test Display

The Test Display shows three types of information.

- Result of each test
- Error messages
- Test report

The Test Display appears as shown in the following illustration.

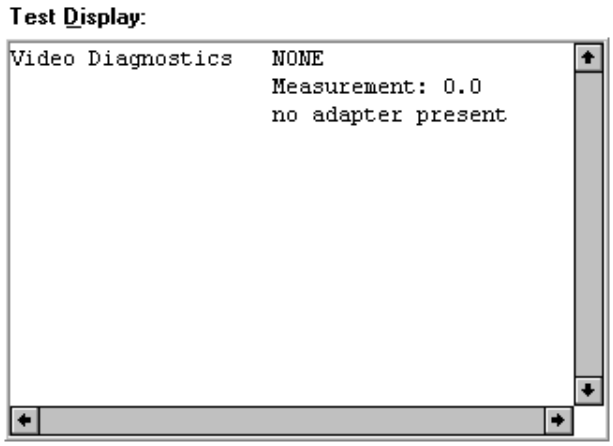

Figure 3-4. Test Display

Result of Each Test

After a test executes, the Test Display shows the complete result of that test. The format of a test result is as follows.

```
Test Name          Result
                   Comment (optional, may be multiple lines)
                   Measurement Comparison  Lower Limit Upper Limit
```

Notice that the number of lines that comprise the test result varies, depending on the type of comparison made and whether or not a test logs a comment. A test result always contains at least one line listing the name and result of the test. The result is the same value shown in the Test Status/Result field of the Sequence Display.

The specific test that logged a comment determines format and content of the Comment line(s). The Measurement field shows the measured value returned by the test. Comparison shows the type of limit checking used to determine whether the test passed. The Condition for Test to Pass column in the following table shows the lower and upper limit values used for pass/fail determination. The possible values of Comparison and their relation to the Lower and Upper Limits (Condition for Test to Pass) appear in the following table.

Table 3-3. Lower and Upper Limits

Comparison	**Condition for Test to Pass**
EQ	Measurement = Lower Limit
NE	Measurement != Lower Limit
GT	Measurement > Lower Limit
LT	Measurement < Upper Limit
GE	Measurement >= Lower Limit
LE	Measurement <= Upper Limit
GTLT	Measurement > Lower Limit and < Upper Limit
GELE	Measurement >= Lower Limit and <= Upper Limit
GELT	Measurement >= Lower Limit and < Upper Limit
GTLE	Measurement > Lower Limit and <= Upper Limit

Error Messages

When the Test Executive detects a run-time error, it displays a description of the error in the Test Display.

The Test Report

You can also view the Test Report in the Test Display. The Test Report records the testing results for the execution of a test sequence. The following segment of monospaced text shows the format of a Test Report.

```
TEST REPORT
Sequence Name:      c:\testexec\examples\computer.squ
Description:        Simulates testing a computer motherboard.
                    Tests Power On, ROM, RAM, Video, and Keyboard.
                    Then runs diagnostics for any areas which fail.
Date:               08-09-1994
Time:               10:48:42
Operator:           John Smith
******************

UUT Serial Number: 1

Power On            PASS
ROM                 PASS
RAM                 PASS
Video               PASS
Keyboard            PASS
ROM Diagnostics     SKIP
RAM Diagnostics     SKIP
Video Diagnostics   SKIP
Keyboard Diagnostic SKIP

UUT Serial Number: 2

Power On            PASS
ROM                 FAIL
RAM                 FAIL
Video               FAIL
Keyboard            FAIL
ROM Diagnostics     NONE
                    Measurement: 5.0
                    Access Error:
                    ROM Bank 5
RAM Diagnostics     NONE
                    Measurement: 5.0
                    Parity Error:
                    RAM Bank 0
Video Diagnostics   NONE
                    Measurement: 5.0
                    no adapter present
Keyboard Diagnostic NONE
                    Measurement: 5.0
                    Keyboard not found
```

Status

The Status indicator, directly below the Test Display, shows the current operating status of the Test Executive. The possible values of the Status indicator and their meanings are as follows.

Table 3-4. Status Indicator Values

Value	Meaning
STOPPED	No test sequence currently running.
RUNNING	Test sequence is running.
LOOPING	Test sequence is looping on a test.

Operator Dialog Boxes

During operation of the Test Executive, several dialog boxes appear that require user action. This section describes these dialog boxes and their actions.

Login

The Login dialog box, shown in the following illustration, prompts the operator for Login Name and Password.

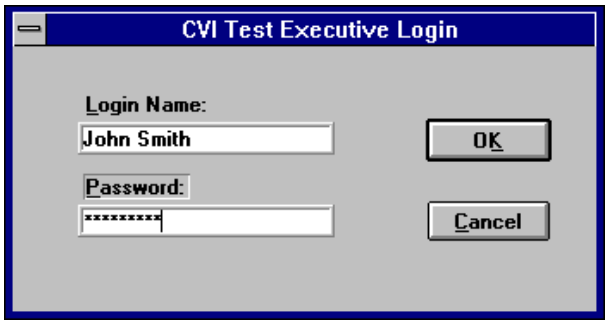

Figure 3-5. Login Dialog Box

Type the appropriate Password to set the desired operating level. The CVI Test Executive dialog box appears when the operator launches the Test Executive and when the operator selects the **Login** menu item from the **File** menu. Click on the **OK** button to confirm the entries made in the Name and Password inputs. Click on the **Cancel** button to remove the dialog box without making any changes to the existing name and password. If you click on **Cancel** in the Login dialog box when the Test Executive first starts running, the Test Executive exits.

UUT Information

The UUT Information dialog box prompts you to enter a serial number for the device to be tested on the next execution of the test sequence.

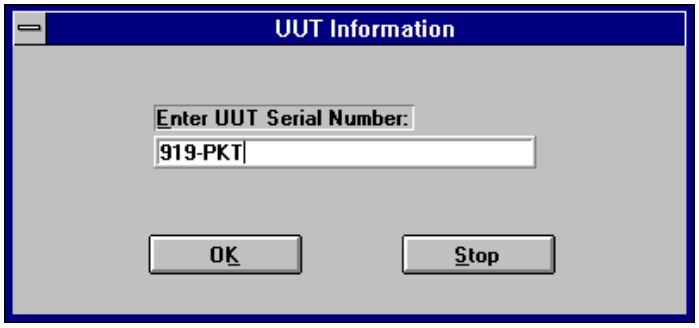

Figure 3-6. UUT Information Dialog Box

The UUT Information dialog box appears only when you use the **Test UUT** button. The serial number can be any ASCII string you choose. Click on the **OK** button to confirm the serial number. Click on **Stop** to stop UUT testing.

Pass, Fail, and Abort Banners

The Pass, Fail, and Abort banners, shown in the following illustrations, indicate whether the current UUT passed or failed. The Pass, Fail, or Abort banner appears at the end of test sequence execution for a single UUT. One of these banners appear only when you initiate testing by clicking on the **Test UUT** button. Press <Enter> or click **OK** to acknowledge the banner and continue testing.

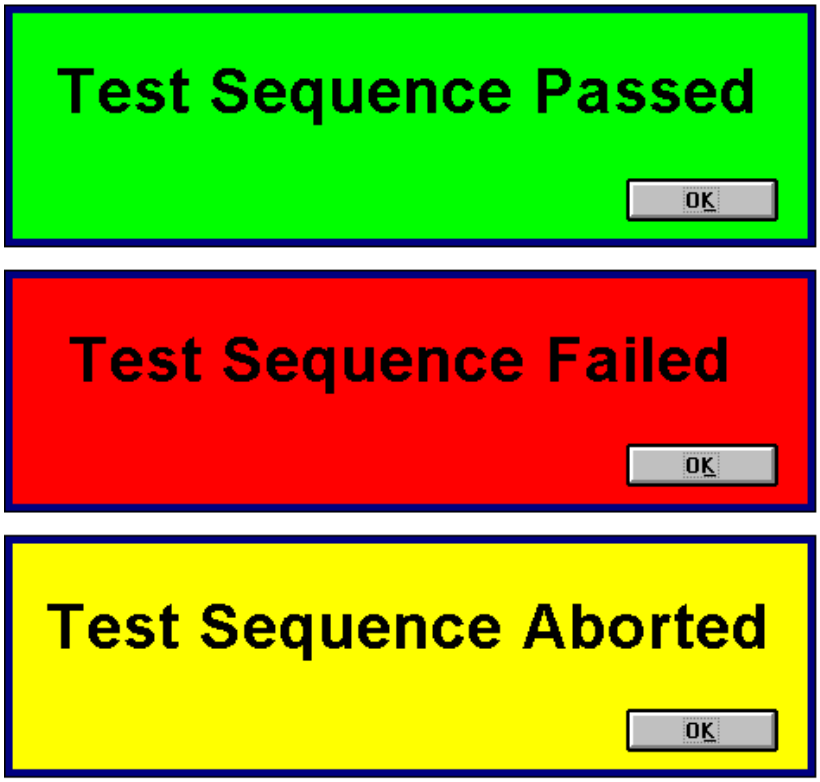

Figure 3-7. The Pass, Fail, and Abort Banners

Run-Time Error Warning

A Run-time Error dialog box appears when a test reports an error. Two types of Run-time Error dialog boxes can appear. The general Run-time Error dialog box is shown in the following illustration.

Figure 3-8. General Run-Time Error Dialog Box

The dialog box shown in the following illustration appears when a run-time error occurs and you have specified a cleanup function for the test sequence. This prompt appears so the operator can choose whether to run the cleanup function.

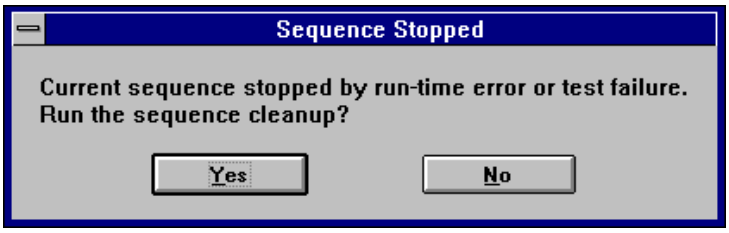

Figure 3-9. Run-Time Error Dialog Box for Cleanup Function

Chapter 4
Creating Tests and Test Sequences

This chapter describes the process of creating new functions and sequences for execution by the Test Executive.

Writing Test Functions

To use the data logging and error reporting capabilities of the Test Executive, you must design tests as C functions with a fixed prototype. You include these functions in your own source code files. You can add your source code files to the project (`testexec.prj`) and then take advantage of the debugging features of LabWindows/CVI. When you finish debugging, you can create object modules from these source code files and remove the source code files from the project file. You can create the object modules in LabWindows/CVI or using the Watcom, Borland, or Symantec 32-bit C compilers. Test functions should have the following prototype:

```
void SampleTest(tTestData *data, tTestError *error);
```

Test Data Structure

A test function uses the Test Data structure to transmit data results that the Test Executive uses to determine if a test passed or failed. The Test Data structure is defined in the following way.

```
typedef struct TestData {
      Status  result;
      double  measurement;
      char    *inBuffer;
      char    *outBuffer;
      char    *modPath;
      char    *modFile;
      void    *hook;
} tTestData;
```

The Test Data structure contains the elements named in the following table.

Table 4-1. Elements of the Test Data Structure

Name	Type	Meaning
PASS/FAIL Flag	`Status`	Set by test function to indicate whether test passed or failed (This flag is observed only if the test has been set to pass or fail based upon a Boolean comparison.)
Measurement	double-precision	Measurement value used for Test Executive Pass/Fail evaluation
inBuffer	string	String passed in by Test Executive
outBuffer	string	String returned by test function
modPath	string	Directory path of file containing test function
modFile	string	File name of file containing test function
hook	generic pointer	Additional user defined data

The Test Executive allocates and frees an input buffer when one is specified for the test. The test function must allocate the **outBuffer** if needed, but the Test Executive frees it. If your test function needs to access another file in its directory (such as a `.uir` file), you can use the **modPath** and **modFile** fields to construct the file name. These fields help you avoid problems if you later move the module containing the test. The **hook** parameter provides a way to pass arbitrary data to the test function. The Test Executive ignores the **hook** parameter.

Test Error Structure

A test function reports errors with the `tTestError` structure. The Error structure is defined in the following way.

```
typedef struct TestError {
      Boolean errorFlag;
      tErrLoc errorLocation;
      int  errorCode;
      char *errorMessage;
} tTestError;
```

The `tTestError` structure contains the following elements.

Table 4-2. Elements of the Error Structure

Name	Type	Meaning
errorFlag	`Boolean`	True if an error occurred, False otherwise
errorLocation	`tErrLoc`	Reserved for Test Executive internal use. Where error occurred (in the test function itself, or one of the setup/cleanup functions)
errorCode	integer	0 if no error, non-zero to indicate specific error
errorMessage	string	Text description of error

The Test Executive uses the contents of the Error structure to determine if a run-time error occurred and takes appropriate action.

Creating Setup and Cleanup Functions

Setup and cleanup functions are special functions for test system configuration, such as turning on a vacuum pump or shutting down power supplies. In general, the setup and cleanup functions always execute, regardless of the status of any test. If the setup or cleanup function encounters a situation that prevents the test sequence from executing, it indicates the condition as a run-time error. Setup and cleanup functions do not log data; they only return run-time error information.

Sample Test Templates

The `Examples` directory contains a collection of sample test templates to help you get started developing your tests. Each test template shows how to use particular features of the test executive, including methods for both Boolean and limit test development, assigning error codes and messages to test operations, passing messages back to the test executive, and using buffer information passed into a test. For new test development, we strongly recommend that you start with one of the test templates in the `Examples` directory to ensure smooth integration into the test executive.

Creating or Editing A Test Sequence

You use the Sequence Editor of the Test Executive to create a test sequence containing your tests. In the Sequence Editor, you can configure the run options and test preconditions that control the flow of your test sequence. You also specify the name of the test function and the file in which it exists.

To create or modify a test sequence, you must choose **Edit Sequence…** from the **Sequence** menu when the Test Executive is at the Developer operating level. This section describes the operation of the Sequence Editor. The following illustration shows the Sequence Editor.

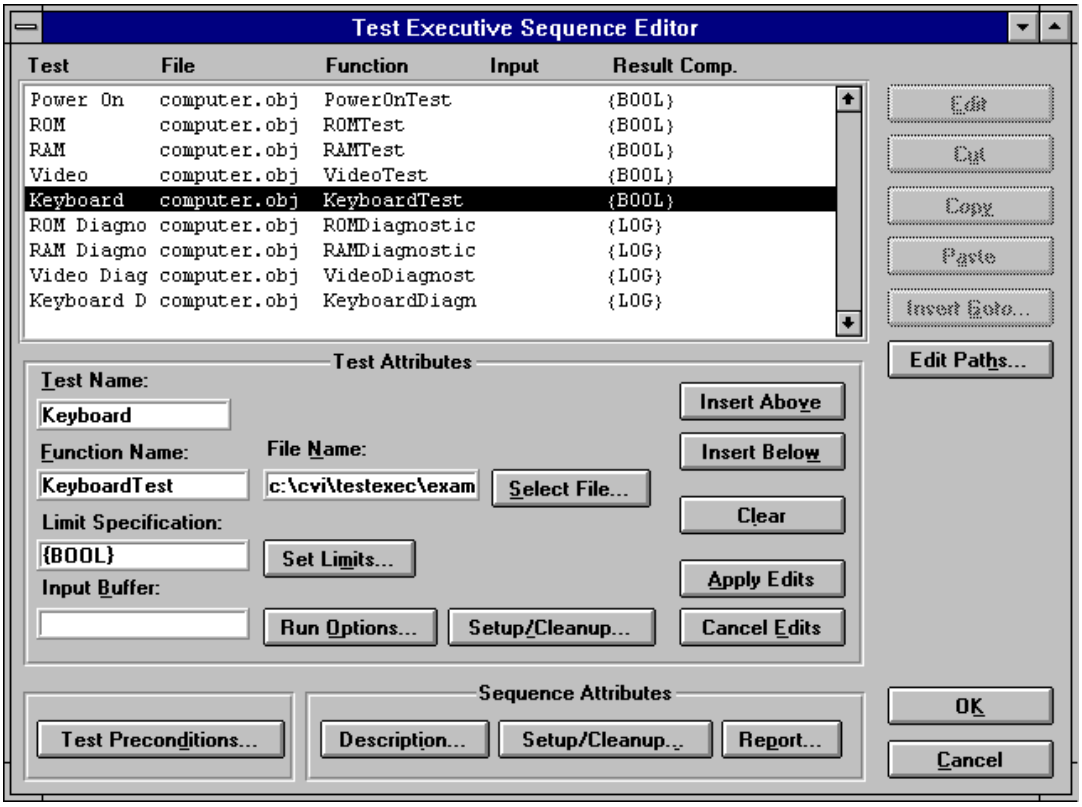

Figure 4-1. Sequence Editor Dialog Box

What Is a Test Sequence?

A test sequence is a collection of data that describes the flow of test execution. The main component of a test sequence is a test. A test is a single execution step in the testing process. A test executes a function to perform the required testing operation. The main components of a test sequence are as follows.

- List of tests
- Setup/Cleanup functions
- Preconditions for flow control based upon Pass/Fail results
- Test report file information
- Description of the sequence

Each of these components is explained in the following sections.

What Is a Test?

A test contains a combination of specifications that tell the Test Executive how to perform a single execution step in the testing process. A test consists of the following specifications.

- Name of test as it appears in the Test column of the Sequence Editor list box
- Function to execute
- Input Buffer containing data for test function
- Limit Specification for determining Pass/Fail status of test
- Run mode specifying whether or not test executes normally
- Fail and Pass Actions (and maximum loop count, if applicable)

Test Editing Operations

The controls grouped directly below the Sequence Editor list box are the test editing area. These controls include Test Name, Function Name, Select File, Set Limit, Run Options, Setup/Cleanup, Insert Below, Insert Above, Replace, and Clear. You use these controls for all operations related to creating or modifying a test. You can add a test, modify a test, copy a test, and delete a test.

Adding a Test

You can insert a new test above or below an existing test in the Test list box. Perform the following steps to add a new test to a sequence.

1. Enter or select the desired values for Test Name, Function Name, Input Buffer, Limit Specification, Run mode, and Setup/Cleanup.
2. Click on the test in the Test list box that is above or below the position where you want to insert the new test.
3. Click on Insert Above or Insert Below to insert the test. If the Test list box contains no tests, the new test appears as the first test in the Test list box.

Modifying a Test

Perform the following steps to modify a test.

1. Click on the test to edit in the Test list box.
2. Click on the **Edit** button. The name of the test you selected now appears in the Test Name string.

3. Enter or select the desired values for Test Name, Function Name, Input Buffer, Run mode, and Setup/Cleanup.

4. Click on the **Apply Edits** button to make the modifications.

Copying a Test

To copy a test, perform the following steps.

1. Click on the test to copy in the Test list box.

2. Click on the **Copy** button.

3. Select the test in the Test list box that is above where you want to put the test you copied.

4. Click on the **Paste** button.

Deleting a Test

To delete a test, perform the following steps.

1. Click on the test you want to delete from the Test list box.

2. Click on the **Cut** button.

Sequence Editor Controls and Indicators

This section describes the controls and indicators on the Sequence Editor front panel.

Edit

The **Edit** button selects the test highlighted in the Test list box for editing. If the highlighted test is a goto, the Edit Goto dialog box appears. Otherwise, the controls in the Test Attributes area are set to the values for the test.

While you edit a test, the **Edit**, **Cut**, **Copy**, **Paste**, and **Insert Goto** buttons are dimmed, meaning you cannot select them. You can use the **Apply Edits** and **Cancel Edits** buttons. The Test list box is locked. This user interface is known as Edit Test mode.

Cut

The **Cut** button copies the test highlighted in the Test list box to the clipboard and deletes the test from the test sequence.

Copy

The **Copy** button copies the test highlighted in the Test list box to the clipboard but does not delete the test from the test sequence.

Paste

The **Paste** button pastes the test in the clipboard below the test highlighted in the Test list box.

Insert Goto

Click on the **Insert Goto** button to open the Insert Goto dialog box. Here, you can add a goto to your test sequence. The following illustration shows the Insert Goto dialog box.

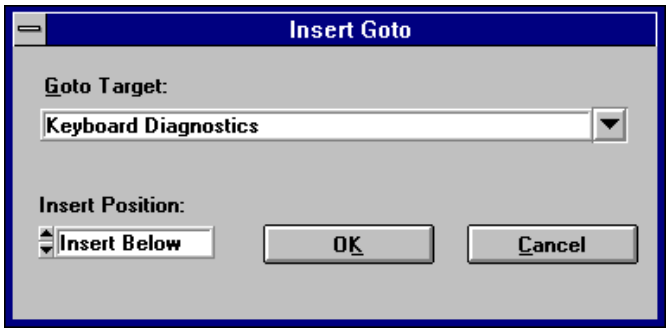

Figure 4-2. Insert Goto Dialog Box

Goto Target

Type the name of the test you want in the Goto Target control. For your convenience, you can click on the arrow to the right of the Goto Target control to see a list of the tests currently defined in the test sequence.

Insert Position

This control determines where the goto is inserted.

Note: ***Goto statements can have preconditions that determine whether they execute, just like a test. You can specify these preconditions in the Preconditions Editor.***

Edit Paths

The **Edit Paths** button opens the following dialog box. Here you can change all the paths stored in the sequence. This helps you update the paths when the sequence changes.

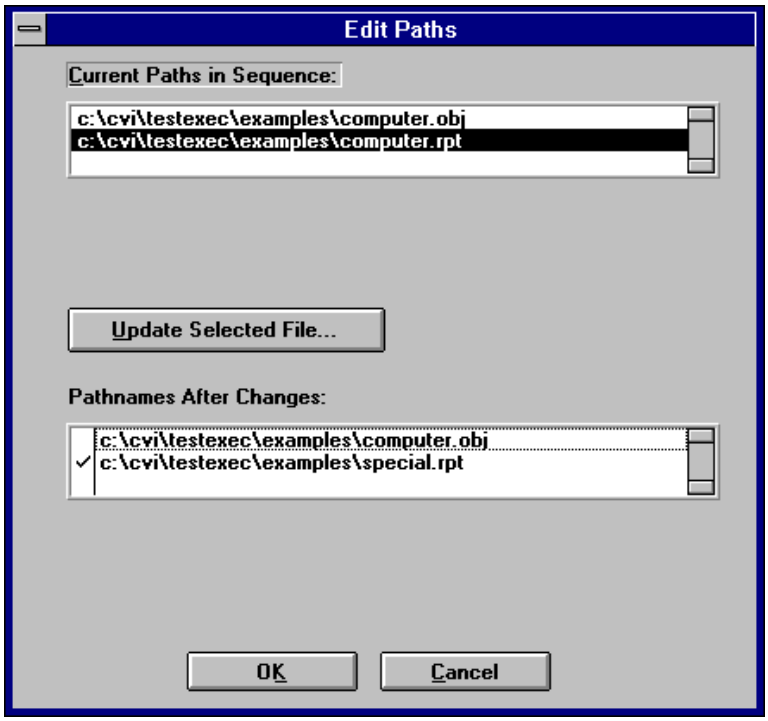

Figure 4-3. Edit Paths Dialog Box

Current Paths in Sequence

This list box show all the unique paths currently in the sequence.

Update Selected File…

Update Selected File... opens a file dialog box where you can specify a new path.

Pathnames after Changes

The Pathnames After Changes list box shows the new pathnames that the Test Executive uses. A check mark appears to the right of the name after a new name has been selected with **Update Selected File...**. The Pathnames After Changes list box is an indicator only.

OK

The **OK** button saves any changes you make to the test sequence, including preconditions, and returns you to the Test Executive main panel.

Note: ***Your changes are not saved to the test sequence file when you click on* OK. *To save the changes to disk, select the* Save *menu item on the Test Executive front panel.***

Cancel

The **Cancel** button discards any changes you make to the test sequence and returns you to the Test Executive main panel.

Test Attributes Area

You use the controls in the Test Attributes area to specify individual tests in the test sequence.

Test Name

Type any ASCII string in the Test Name box. The name appears in the Sequence Display of the Test Executive main panel when the sequence is loaded. Make sure you name each Test Name uniquely.

Function Name

Type the name of the test function in the Function Name box.

File Name

Type the name of the file containing the test into the File Name control. You can also click on the control labeled **Select File…** to open the File dialog box for specifying the file name.

Limit Specification

The Limit Specification specifies the type of limit checking the Test Executive uses to determine if a test passes. You cannot type directly into the Limit Specification indicator. To specify a limit, click on the **Set Limits...** button next to the Limit Specification indicator. Then use the

increment/decrement buttons to set the comparison type and the measurement. The illustration that follows shows the Set Limits dialog box.

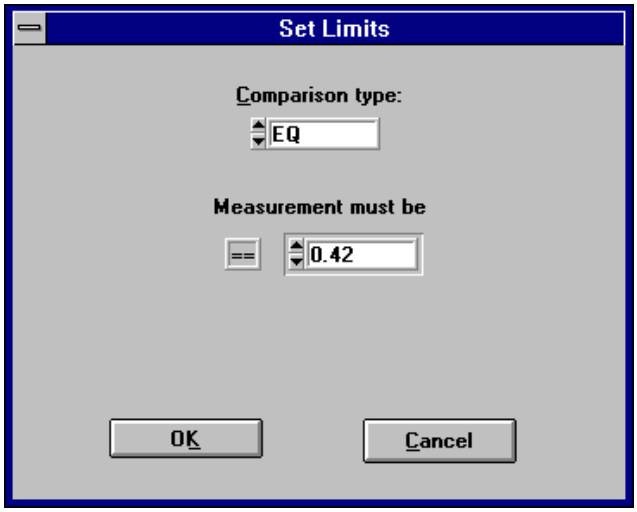

Figure 4-4. Set Limit Dialog Box

Comparison Type specifies the type of comparison to perform, if any, to determine if a test passed. Select the desired limit type from the Comparison Type ring control. The meaning of each value of Comparison Type is as follows.

Table 4-3. Comparison Type Values

Comparison Type	Condition for Test to Pass
EQ	Measurement == Lower Limit
NE	Measurement != Lower Limit
GT	Measurement > Lower Limit
LT	Measurement < Upper Limit
GE	Measurement >= Lower Limit
LE	Measurement <= Upper Limit
GTLT	Measurement > Lower Limit and < Upper Limit
GELE	Measurement >= Lower Limit and <= Upper Limit
GELT	Measurement >= Lower Limit and < Upper Limit
GTLE	Measurement > Lower Limit and <= Upper Limit
BOOL	Measurement not used, test return PASS/FAIL status
LOG	No PASS/FAIL determination–measurement is logged
NONE	No PASS/FAIL determination or data logging

Depending on the setting of Comparison Type, zero, one-limit, or two-limit entry controls appear in the Set Limit Specification dialog box. A one-limit value appears for Comparison Types of `EQ`, `NE`, `GT`, `LT`, `GE`, or `LE`. As shown in the following illustration, two-limit values, a lower and upper limit, appear for Comparison Types of `GTLT`, `GELE`, `GELT`, or `GTLE`. No limits appear for Comparison Types of `BOOL`, `LOG`, or `NONE`.

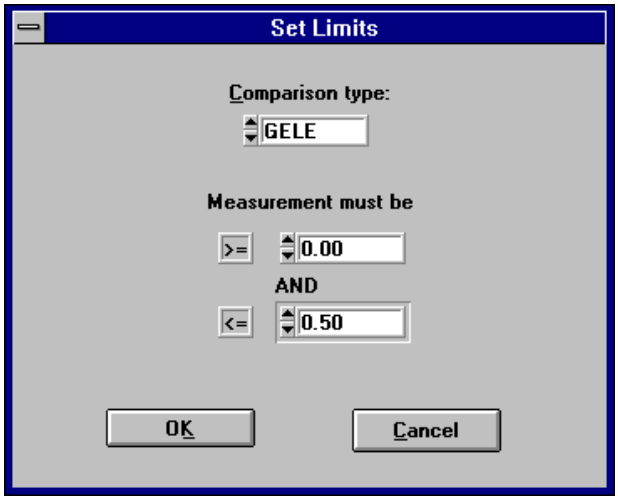

Figure 4-5. Comparison Type Settings

Input Buffer

The Input Buffer is a string that is passed into a test function. The content and meaning of the string are determined by the test function. Enter the desired input string into the Input Buffer control.

Run Options

The **Run Options** button opens the Test Run Options dialog box. Here you can specify the Run Mode, Fail Action, Pass Action, and the maximum number of loops for a test. The Test Run Options dialog box is shown in the following illustration.

Figure 4-6. Test Run Options Dialog Box

Run Mode

Run mode specifies whether the test executes normally. The options for Run mode and their meanings are as follows.

Table 4-4. Run Mode Options

Run Mode	Meaning
Normal	Execute test normally.
Skip	Do not execute the test; set result to SKIP.
Force Pass	Do not execute the test; set result to PASS.
Force Fail	Do not execute the test; set result to FAIL.

Fail Action

Fail Action specifies an action to take when the test fails. The options for Fail Action and their meanings are as follows.

Table 4-5. Fail Action Options

Fail Action	Meaning
Next Test	Continue execution with next test.
Loop	Repeat execution of the test.
Stop	Stop execution of sequence.

Pass Action

Pass Action specifies an action to take when the test passes. The options for Pass Action and their meanings are as follows.

Table 4-6. Pass Action Options

Pass Action	Meaning
Next Test	Continue execution with next test.
Loop	Repeat execution of the test.
Stop	Stop execution of sequence.

Max. Loops

The Max. Loops control appears only when Fail Action or Pass Action is set to `Loop`. Max. Loops specifies the maximum number of loop iterations to perform when, for example, the Fail Action is set to `Loop`, and the test fails.

Setup/Cleanup

The **Setup/Cleanup** button invokes the Setup/Cleanup dialog box for an individual test. The Setup/Cleanup dialog box appears in the following illustration.

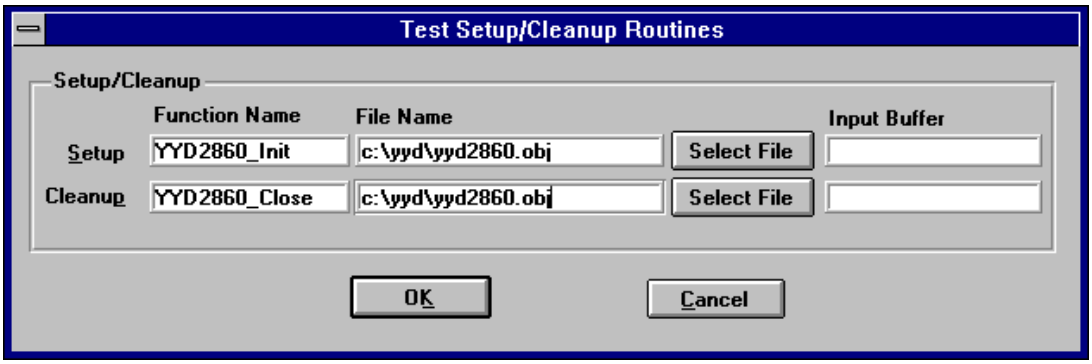

Figure 4-7. Setup/Cleanup Dialog Box

Setup Function

The setup function executes before the test. Enter the name of the setup function in the Function Name control. Enter the name of the file containing the function in the File Name control, or click on the **Select File** button to open the File dialog box, where you can select the file you want. If the Function Name control is left blank under setup, no setup Function runs. (See the *Writing Test Functions* section at the beginning of this chapter for a note about writing your setup function.)

Cleanup Function

The cleanup function executes after the test. Enter the name of the cleanup function in the Function Name control. Enter the name of the file containing the function in the File Name control, or click on the **Select File** button to open the File dialog box, where you can select the file you want. If the Cleanup Function Name control is left blank, no cleanup function runs. (See the *Writing Test Functions* section at the beginning of this chapter for a note about writing your cleanup function.)

Insert Above

The **Insert Above** button inserts the test described in Test Attributes above the test that is selected in the Test list box. If you are working in Edit Test mode, **Insert Above** ends Edit Test mode.

Insert Below

The **Insert Below** button inserts the test described in Test Attributes below the test that is selected in the Test list box. If you are working in Edit Test mode, **Insert Below** ends Edit Test mode.

Clear

The **Clear** button sets all controls in the Test Attributes area to their default values. This button also sets the controls in the Set Limits, Run Options, and Setup/Cleanup dialog boxes to their default values.

Apply Edits

The **Apply Edits** button replaces the test currently selected in the Test list box with the test described in Test Attributes. **Apply Edits** ends Edit Test mode.

Cancel Edits

The **Cancel Edits** button ends Edit Test mode without changes.

Test Preconditions

The **Test Preconditions** button invokes the Precondition Editor. See the *Editing Preconditions* section later in this chapter for more information on editing preconditions.

Sequence Attributes

The controls in the Sequence Attributes area are used to specify the description, setup/cleanup functions, and report file for the test sequence.

Description

The **Description** button displays the dialog box shown in the following illustration, where you can enter and modify the description of the test sequence. This description appears in the Test

Report that is generated when the test sequence is executed. The first line of the description also appears in the Description box on the Test Executive main panel.

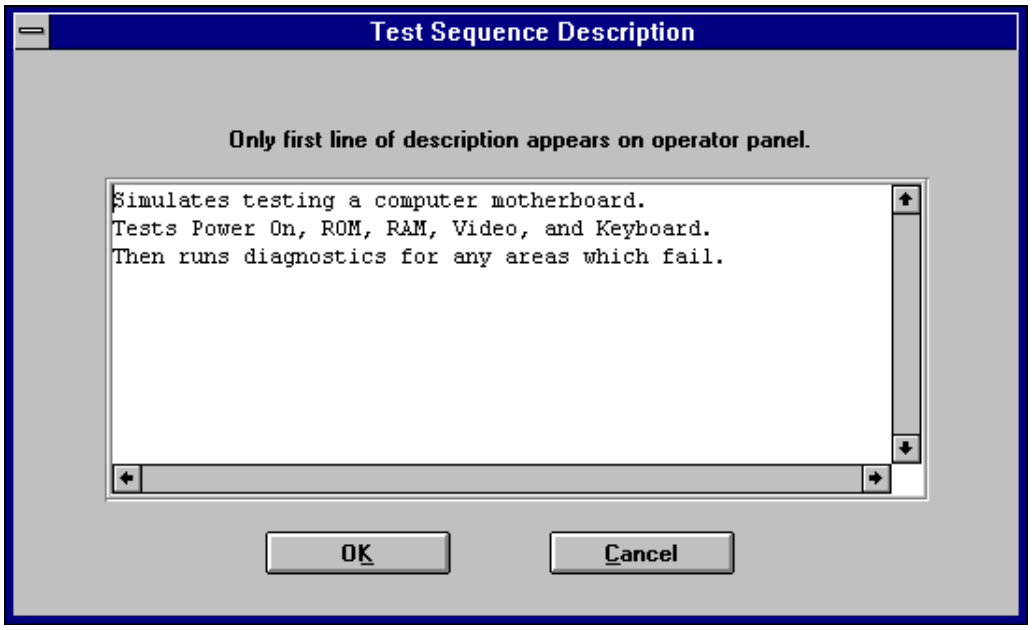

Figure 4-8. Test Sequence Description Dialog Box

Setup/Cleanup

The **Setup/Cleanup** button opens the Sequence Setup/Cleanup Routines dialog box. There are two types of setup/cleanup functions available. Sequence setup/cleanup functions run at the beginning and end of each sequence execution. Sequence load/unload functions run when the sequence is loaded or unloaded. The Sequence Setup/Cleanup Routines dialog box appears in the following illustration.

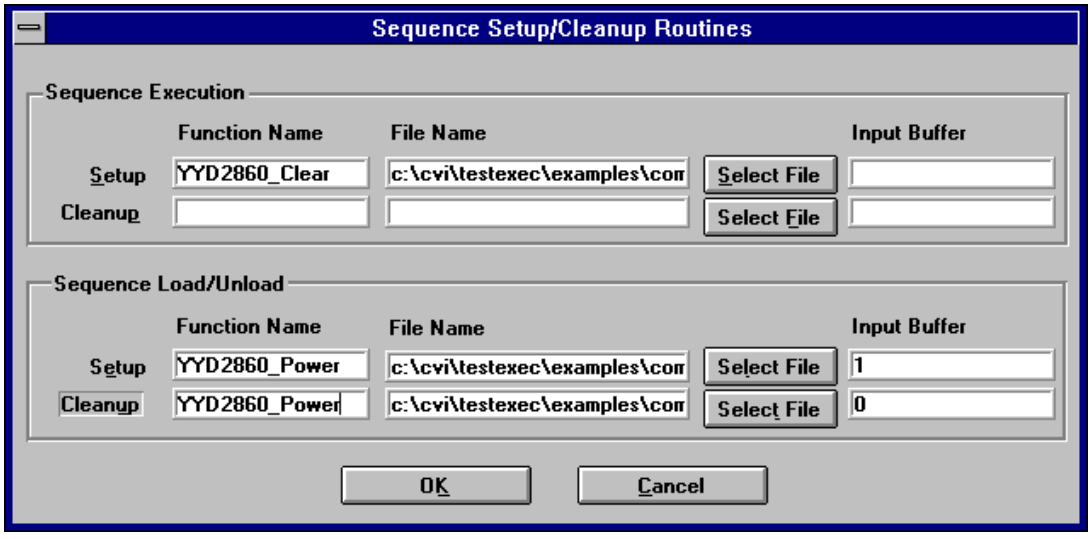

Figure 4-9. Sequence Setup/Cleanup Dialog Box

Report…

You click on the **Report…** button to set the attributes of your Test Report. Type the name of the report file you want to create in the Test Report File control. You can click on the **Select File…** button to open the File dialog box, where you can select the name of the report. When you set Report File Mode to **Append** the Text Executive will add your report to the end of an existing report. Choose **Overwrite** to replace the existing report file. Select Lock File Name to prevent users from changing the name of the report file. The Set Default Report File dialog box appears in the following figure.

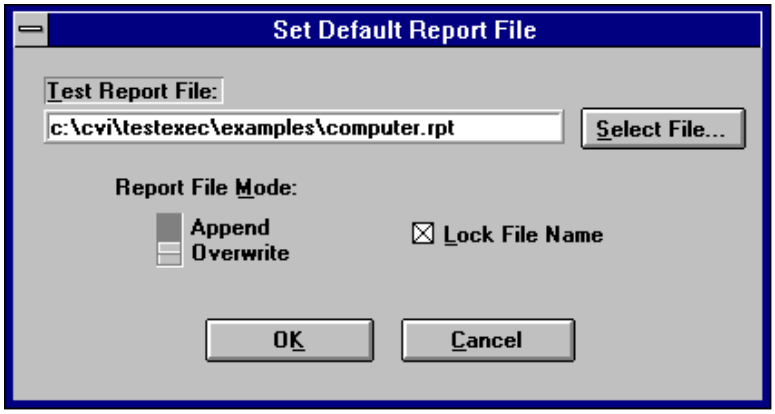

Figure 4-10. Set Default Report File Dialog Box

Editing Preconditions

The preconditions of a particular test specify what other tests must pass or fail before that particular test executes. You can use the Precondition Editor shown in the following figure to define the preconditions of tests.

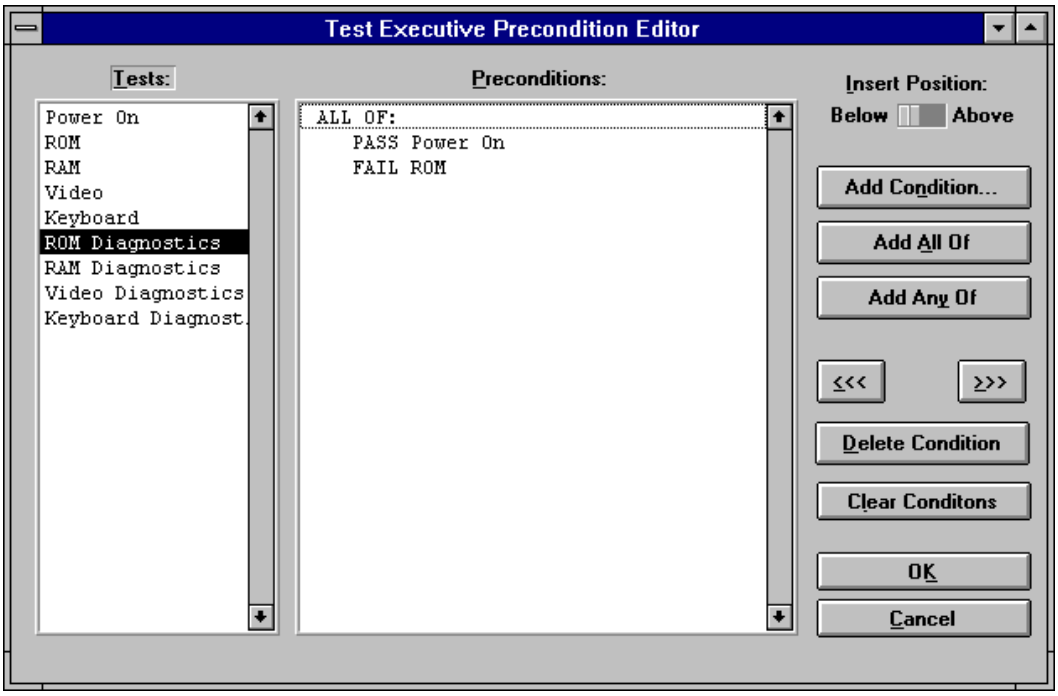

Figure 4-11. Precondition Editor

The Precondition Editor shows preconditions for a selected test. In the Tests list box, the name of each test in the test sequence appears. One of the test names in the Tests list box is always highlighted. The Preconditions list box shows the precondition tests—those tests on which test execution depend—for any test name you select. In the illustration above, Power On test and ROM test are the preconditions for ROM Diagnostics. ROM Diagnostics only runs when Power On test passes and ROM test fails.

Take the following steps to specify the preconditions for a test.

1. Click in the Tests list box to select a test. If you want multiple preconditions, group them under `ALL OF:` or under `ANY OF:`.

   a. Click on **Add All Of** when all of the preconditions you specify must be true.

   b. Click on **Add Any Of** when at least one of the preconditions must be true.

   c. You can also nest `ALL OF:` and `ANY OF:` to create more complex preconditions.

2. Click on **Add Condition…**.

In the Add Condition dialog box, set the Type switch to PASS or FAIL, depending on whether you want the precondition test to pass or fail. Then select the precondition test from the Tests list box, and click on **OK**. You can select multiple precondition tests from the Tests list box. The setting of the Type switch determines the pass or fail setting of the next precondition test that you select. The letter `P` beside a test name indicates a pass setting.
`F` indicates a fail setting.

3. Continue adding conditions until you complete the desired settings.

You can use the other controls on the Precondition Editor as follows.

- The Insert Position switch determines whether new preconditions will be inserted before or after the current precondition.
- The **Add All Of** button inserts `ALL OF:` to begin a block of preconditions which must all be true.
- The **Add Any Of** button inserts `ANY OF:` to begin a block of preconditions of which at least one must be true.
- The **Add Condition…** button brings up the Add Condition dialog box, shown in the following illustration.

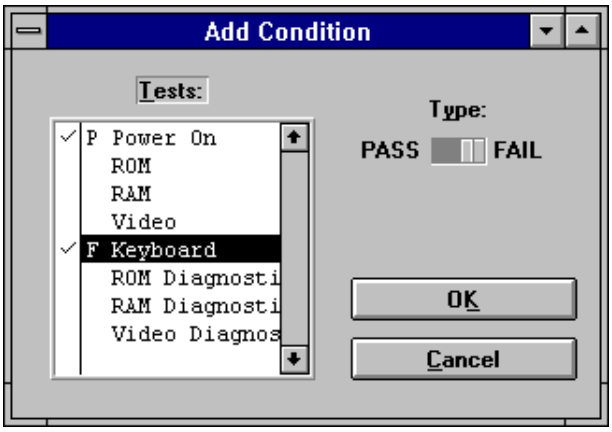

Figure 4-12. Add Condition Dialog Box

- The Tests list box shows the available precondition tests. The Type switch allows you to specify whether the precondition test must pass or fail.

- The **Move to the left** and **Move to the right** buttons (shown in the following figure) allow you to adjust the level of the currently selected precondition. In general, **Add Condition** sets the level properly, so you should seldom have need of the **Move to the left** and **Move to the right** buttons.

Figure 4-13. Move to the Left Button and Move to the Right Button

- The **Delete Condition** button deletes the selected precondition.
- The **Clear Conditions** button clears all of the preconditions for the test selected in the Tests list box.
- The **OK** button saves any changes made to the test sequence preconditions and returns you to the Sequence Editor.

Changes you make to preconditions take effect only when you click on the **OK** button in the Sequence Editor. All preconditions are saved with the test sequence when you select **Save** from the menu on the Test Executive main panel to save the test sequence to disk.

The **Cancel Edits** button discards any changes made to the test sequence preconditions and returns you to the Sequence Editor.

Relationship between Preconditions, Run Mode, and Test Flow

The preconditions and run mode for each test determine the test execution flow for a test sequence. The Test Executive performs the following steps to determine whether or not to execute a given test.

1. The Text Executive evaluates the preconditions for the test. For a test to execute, the result of each precondition test must match the result specified in the preconditions. If evaluation of preconditions indicates that the current test should be skipped, the test result is set to `SKIP`.

2. The Test Executive checks the Run mode of the test. If the Run mode is `Normal`, the test executes. If Run mode is set to any value except `Normal`, the test does not execute. Table 4-7 shows the value that each Run mode generates for a skipped test.

Table 4-7. Run Mode Test Result Values

Run Mode	**Value Generated**
Skip	`SKIP`
Force PASS	`PASS`
Force FAIL	`FAIL`

When the Text Executive evaluates preconditions, it does not distinguish between a real `PASS` or `FAIL` result—where the test actually executed—and a forced `PASS` or `FAIL` result.

Chapter 5
Modifying the Test Executive

This chapter describes the internal structure of and how to modify the Test Executive. When you need to go beyond the standard Test Executive functions, you can refer to the information in this chapter. This chapter explains the organization of the Test Executive so that you can change the behavior of the application when you need to. The chapter covers the following topics.

- Organization of source code files
- Common modifications
- Advanced modifications
- Writing a test executive

Organization of Source Files

The Test Executive contains several source code files. The following table lists the name and use of each file. More detailed descriptions of each file appear after the table.

Table 5-1. List of Files in the Test Executive

File Name	File Use
`txmain.c`	Test Executive Main Panel
`txedseq.c`	Sequence Editor
`txedpc.c`	Precondition Editor
`txlogin.c`	Login Panel
`txprecnd.c`	Evaluate, load, save preconditions
`txreport.c`	Create and save report
`txsavres.c`	Maintain history of test results
`cvitxuir.uir`	User Interface
`txengine.c`	Maintain, load, and save sequences; run tests

`txmain.c` is the main source code file for the Test Executive. In addition to controlling the other modules, `txmain.c` handles the Text Executive front panel. The function `main()` handles initialization and starts the user interface. Thereafter, the Test Executive makes extensive use of callbacks. The important callbacks in `txmain.c` are `MainPanelCallback` and `MainMenuCallback` which handle all the menus and controls on the front panel. A major task of `txmain.c` is running tests. In particular, `RunTestUUT`, `SinglePass`, `LoopTest`, and `RunSingleTest` are functions in `txmain.c` that handle the running of tests.

`txedseq.c` handles the Sequence Editor panel and its popup dialog boxes. The function SetupEdit starts the Sequence Editor. The main callback for the Sequence Editor is `SeqEdCallback`. Each pop-up dialog box also has a callback: `GotoCallback`, `RunOptionCallback`, `ReportCallback`, `LimitCallback`, `TestPrePostCallback`, `SeqPrePostCallback`, `DescrCallback`, or `EdPathsCallback`. The most important global variable in `txedseq.c` is the array `gEditData` which contains the specification for each test in the sequence.

`txedpc.c` handles the Precondition Editor panel and its pop-up dialog boxes. The function `PE_SetupPreCondEd` starts the Precondition Editor. The main callback for the Precondition Editor panel is `PreCondEdCallback`. The Add Condition dialog box also has the callback `AddConditionCallback`. The array variable `sAllTestPCArray` contains the preconditions for all tests.

The function `OpenLogin` in `txlogin.c` handles login by users. Most importantly, `txlogin.c` sets the global variable `gLoginLevel` which determines whether the Test Executive is in Operator, Technician, or Developer operating level.

`txprecnd.c` handles loading, saving, and evaluation of preconditions. The functions for saving and loading preconditions are `PC_Save` and `PC_Load`. `PC_EvalPreconditions` evaluates the preconditions of a test. The array variable `sSeqPreConds` contains the preconditions.

`txreport.c` handles test report generation. The major functions are `WriteRptHeader`, `WriteSeqHeader`, and `WriteTestResult`. Each string written to the report file also becomes part of the global array `gTestReport` for use by the **View** command in the **Report** menu. When `gTestReport` is full, the system discards the oldest strings.

`txsavres.c` maintains a record of current and previous test results. The Test Executive uses the results for the current sequence when evaluating preconditions. Previous results go into a history list as a starting point for users who want to add analysis of test results. The function `SR_SaveOneResult` stores the results of one test with results for the rest of the sequence. `SR_RememberCurrResults` copies the results for the most recent sequence execution to the history list. `SR_ForgetCurrResults` discards the most recent sequence results. The array `currResults` stores the current results and the array `seqResults` stores the history.

`cvitxuir.uir` contains the specifications for the user interface of the Test Executive.

`txengine.c` contains the functions that maintain, load, and save sequences, and run tests.

Common Modifications

This section describes common modifications that you can make to the Test Executive. This section covers the following areas.

- Built-in Customization
- Passwords
- Pass, Fail, and Abort banners
- UUT Serial Number prompt
- Test report
- Standalone executables

Built-In Customization

Several customization options are already included in the Test Executive. By defining or undefining C language macros in the file `cvitxcst.h` you can activate or deactivate these options. See `cvitxcst.h` for more information.

Changing Passwords

You can set the passwords that determine operating level (Developer, Technician, and Operator) according to your specifications by modifying the function `CheckLogin`, found in `txlogin.c` (the *Operating Levels* section in Chapter 1, *Introduction*, describes the operating modes). Passwords in `CheckLogin` are case sensitive.

Changing Pass, Fail, and Abort Banners

You can change the Pass, Fail, and Abort banners to display a banner that you design. On a color monitor, the Pass, Fail, and Abort banners have a colored background (green for Pass, red for Fail, yellow for Abort), an **OK** button, and a large label containing the word Pass, Fail, or Abort. These banners are located in `cvitxuir.uir` and you can change them with the User Interface Editor.

Changing the UUT Serial Number Prompt

You can also modify the panel that asks for the UUT serial number, which is located in `cvitxuir.uir`. If you add controls to this panel, you need to modify the function `RunTestUUT` in `txmain.c`.

You can also make other modifications to the serial number prompt, such as adding a routine that reads the serial number from a bar code reader through an RS-232 port. Again, to make this change you need to modify the function `RunTestUUT`.

Changing the Test Report

The `txreport.c` module generates the Test Report. The following illustration shows a standard Test Report. You can modify the test format of the Test Report to suit your needs.

```
TEST REPORT
Sequence Name:      c:\testexec\examples\computer.squ
Description:        Simulates testing a computer motherboard.
                    Tests Power On, ROM, RAM, Video, and Keyboard.
                    Then runs diagnostics for any areas which fail.
Date:               08-09-1994
Time:               10:48:42
Operator:           John Smith
******************

UUT Serial Number: 1

Power On            PASS
ROM                 PASS
RAM                 PASS
Video               PASS
Keyboard            PASS
ROM Diagnostics     SKIP
RAM Diagnostics     SKIP
Video Diagnostics   SKIP
Keyboard Diagnostic SKIP

UUT Serial Number: 2

Power On            PASS
ROM                 FAIL
RAM                 FAIL
Video               FAIL
Keyboard            FAIL
ROM Diagnostics     NONE
                    Measurement: 5.0
                    Access Error:
                    ROM Bank 5
RAM Diagnostics     NONE
                    Measurement: 5.0
                    Parity Error:
                    RAM Bank 0
Video Diagnostics   NONE
                    Measurement: 5.0
                    no adapter present
Keyboard Diagnostic NONE
                    Measurement: 5.0
                    Keyboard not found
```

Several routines control the formatting of the report: `WriteRptHeader`, `WriteSeqHeader`, `WriteTestResult`, `WriteStopReason`, and `WritePrePostOutBuffer`.

Standalone Executables

The Test Executive Toolkit is shipped with a standalone executable of the Test Executive. You may want to create your own standalone executable for any modifications you may make.

You can also create a standalone executable to run a particular sequence by placing the full path of the desired sequence file on the command line. If your standalone executable is intended for operator use only, you can modify the function `OpenLogin` in `txlogin.c` to always set the login level to operator.

Please refer to your LabWindows/CVI documentation for information about standalone executables and the command line.

Advanced Modifications

The following section tells you how to make advanced modifications to the Test Executive: changing preconditions and writing your own test executive.

Replacing the Preconditions

You may want to replace the precondition processor. For example, you might prefer that each test have a Pass `goto` and a Fail `goto`. The steps you must take to replace the precondition processor are relatively straightforward, for the following reasons.

- The test executive application calls the precondition module infrequently, so you can quickly locate all the calls.
- The preconditions are appended to the end of the `.squ` file rather than being mixed in with the rest of the test sequence.
- The functions that save and load preconditions are not in `txengine.c`, so you can change them without changing `txengine.c`.

To make replacements you need to make the following changes.

- Write replacement modules for `txedpc.c` and `txprecnd.c`
- Replace precondition panels in `cvitxuir.uir`.
- Change the function `RunSequence` in `txmain.c` to use your precondition replacement.
- Change `txedseq.c` to match your precondition replacement.
- Change calls to `TX_SaveSequence` and `TX_LoadSequence` to pass your save and load functions rather than `PC_Save` and `PC_Load`.

Writing a Test Executive

The Test Executive Engine contains the support functions needed to write a test executive. Chapter 6, *Function Descriptions for the Test Executive Engine*, describes the available functions. The `tinyexec` directory contains two example programs, `tiny_bld.prj` and `tiny_run.prj`, which illustrate the basic steps for writing a test executive. `tiny_bld` shows you how create a sequence and `tiny_run` shows you one way to make a sequence execute. For the sake of simplicity, both examples omit most error checking, ignore preconditions, ignore pre/post tests, and support only the `BOOL` Limit Specification. When you develop a Test Executive, you can add such features as needed.

Chapter 6
Function Descriptions for the Test Executive Engine

This chapter describes the functions in the LabWindows/CVI Test Executive engine. The section called *Test Executive Engine Overview* contains general information about the Test Executive engine functions. The *Test Executive Engine Function Reference* section presents function descriptions in alphabetical order.

Test Executive Engine Overview

The Test Executive source code file contains a library of functions that perform many low-level operations to maintain, load, and save sequences, and run tests. This library of functions is called the Test Executive *engine*.

You can modify the Test Executive by editing the source code built on top of the Test Executive engine library of functions. If you need to make fundamental changes to the application itself, you can develop your own test executive application from scratch, using as a starting point the Test Executive engine functions described in this chapter. Do not modify the functions in the Test Executive engine library functions unless you are an experienced C programmer with thorough knowledge of test executive systems.

The Test Executive Engine Function Panels

The Test Executive engine function panels are grouped in the following tree structure according to the types of operations they perform.

Table 6-1. Test Executive Engine Function Tree

Test Executive Engine	
Initialization/Reset	
Restart Test Executive	***TX_RestartEngine***
Edit Test Sequence	
Add Sequence Pre or Post Test	***TX_AddPrePostTest***
Add Test After Current Test	***TX_AddTestNext***
Add Test Before Current Test	***TX_AddTestPrevious***
Cut Test From Sequence	***TX_CutTest***
Paste Test After Current Test	***TX_PasteTestNext***
Paste Test Before Current Test	***TX_PasteTestPrev***
Verify Test Sequence	
Verify Test	***TX_VerifyTest***
Verify Sequence Pre/Post Test	***TX_VerifyPrePostTest***
Verify Sequence	***TX_VerifySequence***
Manipulate Current Test Pointer	
Set the CTP to a Test	***TX_SetCurrTest***
Move the CTP to the Next Test	***TX_NextTest***
Move the CTP to the Previous Test	***TX_PrevTest***
Read/Modify Current Test	
Get Information from the Engine	***TX_GetEngineInfo***
Change Settings in Engine	***TX_SetEngineInfo***
Run Tests	
Run Current Test	***TX_RunTest***
Run Sequence Pre/Post Test	***TX_RunPrePostTest***
Load/Save Sequence	
Load a Sequence from Disk	***TX_LoadSequence***
Save a Sequence to Disk	***TX_SaveSequence***
Load/Save Support	
Read Byte	***TX_ReadByte***
Read Short Integer	***TX_ReadShort***
Read Integer	***TX_ReadInt***
Read Double	***TX_ReadDouble***
Read String	***TX_ReadString***
Write Byte	***TX_WriteByte***
Write Short Integer	***TX_WriteShort***
Write Integer	***TX_WriteInt***
Write Double	***TX_WriteDouble***
Write String	***TX_WriteString***
Utility	
Get Base File Name	***TX_GetBaseFileName***
Get File Directory	***TX_GetFileDir***
Duplicate String	***TX_StrDup***

The bold headings on the left side of the tree are the names of function classes and subclasses. Function classes and subclasses contain groups of related function panels. The subheadings in plain text are the names of individual function panels. Each User Interface Library function panel generates one function call. The name of each function call is to the right of its corresponding function panel name in bold, italic text.

Descriptions of the function classes appear in the following list.

- **Initialization/Reset** functions initialize the Test Executive Engine.
- **Edit Test Sequence** functions add and remove tests from the current sequence.
- **Verify Test Sequence** functions load tests from external modules and verify that the tests can be run.
- **Manipulate Current Test Pointer** functions allow changing the Current Test Pointer (CTP).
- **Read/Modify Engine Information** functions access sequence and test information stored in the Test Executive Engine.
- **Run Tests** functions run tests.
- **Load/Save Sequence** functions load or save sequences.
- **Utility** functions provide utility routines.

Include Files

The Test Executive Engine provides an include file that contains function declarations and defined constants for all of the library routines. This include file, named `txengine.h`, must be in all code modules that reference the Test Executive Engine.

Reporting Errors

Most of the functions in the Test Executive Engine return an integer code containing the result of the call. If the return code is negative, an error occurred. If the return code is greater than or equal to zero, the function completed successfully. See Appendix A, *Error Codes and Engine Constants*, for a list of error codes.

Test Executive Engine Function Reference

This section describes each function in the Test Executive Engine. The functions are arranged alphabetically.

TX_AddPrePostTest

```
int status = TX_AddPrePostTest (char *testName, char *testDescription,
                                char *fileNameforTestModule,
                                char *functionNameofTest,
                                int executionTime, int testType );
```

Purpose

This function adds a pre- or post-test to the Test Executive Engine. You may specify that the test run every time the sequence executes or that it run only when the system loads or unloads the sequence.

Parameters

Input	**testName**	string	Name that uniquely identifies the test
	testDescription	string	Description that identifies the test more completely
	fileName	string	Path and file name for the module that contains the test function
	functionName	string	Name of actual function in the module, specified by the previous parameter
	executionTime	integer	If non-zero, the test runs only when the sequence is loaded or unloaded. If zero, the test runs every time the sequence is executed.
	testType	integer	If non-zero, the test is a pre-test. If zero, the test is a post test.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_AddTestNext

```
int status = TX_AddTestNext (char *testName, char *testDescription,
                             char *testFileName, char *testFunctionName,
                             char *preTestFileName, char *preTestFunctionName,
                             char *postTestFileName, char *postTestFunctionName);
```

Purpose

This function adds a test function into the test sequence. `TX_AddTestNext` adds the text function after the test that the Current Test Pointer (CTP) points to. `TX_AddTextNext` then increments the CTP.

Parameters

Input	**testName**	string	Name that uniquely identifies the test
	testDescription	string	Description that identifies the test more completely
	testFileName	string	Path and file name for the module that contains the test function
	testFunctionName	string	Name of the test function
	preTestFileName	string	Path and file name of the module that contains the local pre-test function
	preTestFunctionName	string	Name of the local pre-test function
	postTestFileName	string	Path and file name of the module that contains the local post-test function
	postTestFunctionName	string	Name of the local post-test function

Return Value

status	integer	Refer to Appendix A for error codes.

TX_AddTestPrevious

```
int status = TX_AddTestPrevious(char *testName, char *testDescription,
                                char *testFileName, char *testFunctionName,
                                char *preTestFileName, char *preTestFunctionName,
                                char *postTestFileName,
                                char *postTestFunctionName);
```

Purpose

This function adds a test function into the test sequence. `TX_AddTestPrevious` adds the Test Function before the test that the CTP points to. `TX_AddTestPrevious` then decrements the CTP.

Parameters

Input	**testName**	string	Name given to the test to more descriptively identify the test
	testDescription	string	Description given to the test to more descriptively identify the test
	testFileName	string	Path and file name for the module that contains the test function
	testFunctionName	string	Name of the test function
	preTestFileName	string	Path and file name of the module that contains the local pre-test function
	preTestFunctionName	string	Name of the local pre-test function
	postTestFileName	string	Path and file name of the module that contains the local post-test function
	postTestFunctionName	string	Name of the local post-test function

Return Value

status	integer	Refer to Appendix A for error codes.

TX_CutTest

`int` **TX_CutTest** `(void);`

Purpose

This function will remove the test pointed to by the Current Test Pointer and store it in an internal clipboard. You may reinsert the test using `TX_PasteTestNext` or `TX_PasteTestPrev`.

Note: ***When you cut a test and the clipboard already holds a test, then the previous test is lost.***

Return Value

status	integer	Refer to Appendix A for error codes.

TX_GetBaseFileName

`char *`**baseFileName** = **TX_GetBaseFileName** `(char *`**fullPathName**`);`

Purpose

This function returns a copy of just the file name from the full path name of a file.

Parameters

Input	**fullPath**	string	The full path name of a file

Return Value

baseFileName	string	A copy of the file name found in the full path name

---

TX_GetEngineInfo

`int` **status** = **TX_GetEngineInfo** `(unsigned short` **field**`, void *`**newValue**`);`

Purpose

Returns information stored in the Engine and/or in the test that the Current Test Pointer points to.

Notice that the data returned is only a copy of the actual data in the engine. Modifying the return value does not affects the data inside the engine.

Parameters

Input	**field**	short integer	Indicates what information to get. Refer to Appendix A for valid fields.
Output	**newValue**	void	Pointer to returned information. Caller responsible for freeing memory used by this value.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_GetFileDir

`char *`**fileDirectory** = **TX_GetFileDir**`(char *`**fullPathName**`);`

Purpose

This function returns a copy of the directory from the full path name of a file.

Parameters

Input	**fullPath**	string	The full path name of a file

Return Value

fileDirectory	string	A copy of the directory name for a file

TX_LoadSequence

`int` **status** = **TX_LoadSequence** `(char *`**fileName**, `void *`**specialLoadFunction**`);`

Purpose

Loads a test sequence.

Parameters

Input	**fileName**	string	The name (and drive/path) for the file containing the test sequence to be loaded
	specialLoadFunction	function pointer	Points to a function for loading additional information. See *Parameter Discussion.*

Return Value

status	integer	Refer to Appendix A for error codes.

Parameter Discussion

specialLoadFunction allows you to set up a function to load additional data that the Test Executive Engine did not load. For example, the Test Executive normally uses this function to load the preconditions. This function allows you to add extensions without having to modify the engine itself. This special load function has the following prototype:

`int` **SpecialLoadFunc**`(int` **fileHandle**, `int` **numTests**`)`

fileHandle is the handle of the open sequence file and **numTests** is the number of tests in the sequence.

TX_NextTest

`int` **status** = **TX_NextTest**`(void);`

Purpose

Moves the Current Test Pointer (CTP) to the next test in the test sequence. If the CTP is currently at the end of the list, the CTP does not change, but an error code is returned.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_PasteTestNext

`int` **status** = **TX_PasteTestNext**`(void);`

Purpose

This function inserts the test that is stored in the clipboard after the test that the CTP points to.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_PasteTestPrev

`int` **status** = **TX_PasteTestPrev**`(void);`

Purpose

This function inserts the test that is stored in the clipboard before the test that the CTP points to.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_PrevTest

`int` **status** = **TX_PrevTest** (`void`);

Purpose

Moves the Current Test Pointer (CTP) to the previous test in the test sequence. If the CTP is currently at the beginning of the list, the CTP remains there the function returns an error code.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_ReadByte

`int` **status** = **TX_ReadByte** (`int` **fileHandle**, `char *` **byteRead**);

Purpose

This function reads a byte value from the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file from which the function reads the value
Output	**byteRead**	pointer to `char`	Pointer to the byte value that the function reads from the file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_ReadDouble

`int` **status** = **TX_ReadDouble**(`int` **fileHandle**, `double *`**doubleRead**);

Purpose

This function reads a double-precision value from the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file from which the function reads the value
Output	**doubleRead**	pointer to double-precision	Pointer to the double-precision value that the function reads from the file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_ReadInt

`int` **status** = **TX_ReadInt**(`int` **fileHandle**, `int *`**intRead**);

Purpose

This function reads a integer value from the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file from which the function reads the value
Output	**intRead**	integer	Pointer to the integer value the function reads from the file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_ReadShort

`int` **status** = **TX_ReadShort**(`int` **fileHandle**, `int *`**shortRead**);

Purpose

This function reads a short integer value from the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file from which the function reads the value
Output	**shortRead**	short	Pointer to the short integer value the function reads from the file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_ReadString

`char *`**theString** = **TX_ReadString**(`int` **fileHandle**, `int *`**status**);

Purpose

This function reads a string from the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file from which the function reads the value
Output	**status**	integer	Refer to Appendix A for error codes.

Return Value

theString	char	The string read from the file.

TX_RestartEngine

`int` status = **TX_RestartEngine** (`void`);

Purpose

This function initializes the Test Executive Engine. If there is a test sequence currently in the Engine when you call this function, that sequence is deleted.

Note: ***This function does not warn you that there is a sequence in the Engine. You must save the test sequence before calling this function. See*** `TX_SaveSequence` ***for more information.***

Return Value

status	integer	Refer to Appendix A for error codes.

TX_RunPrePostTest

`int` **status** = **TX_RunPrePostTest** (void ***testData**, void ***testError**, `int` **executionTime**, `int` **testType**);

Purpose

Runs a test prior to or after a sequence. Parameters specify that the test run every time the sequence runs or that it run only when the system loads or unloads the sequence. The parameters also set up the test as a pre- or post-test.

Parameters

Input/ Output	**testData**	tTestData (See *Index*.)	Pointer to structure containing the test data. See *Parameter Discussion*.
	testError	tTestError (See *Index*.)	Pointer to structure returning error information. See *Parameter Discussion*.
Input	**executionTime**	integer	If non-zero, the test runs only when the sequence is loaded or unloaded. If zero, the test runs every time the sequence is executed.
	testType	integer	If non-zero, the test is a pre-test. If zero, the test is a post-test.

Return Value

status	integer	Refer to Appendix A for error codes.

Parameter Discussion

Sequence pre- and post-tests can ignore the **testData** and **testError** structures.

TX_RunTest

`int` **status** = **TX_RunTest** (void ***testData**, void ***testError**, `int` ***prePostTestError**);

Purpose

Runs the test that the CTP points to. If a pre-test exists, it executes first. After the pre-test runs, or if there is no pre-test, the test executes. Finally, the post-test executes.

Parameters

Input/ Output	**testData**	tTestData (See *Index*.)	Pointer to structure containing the test data. This pointer will be passed to the test function. See *Parameter Discussion.*
	testError	tTestError (See *Index*.)	Pointer to a structure for returning error information from the test function. See *Parameter Discussion* for more information.
Output	**prePostTestError**	integer	Flag indicating an error occurred in the pre- or post-test

Return Value

status	integer	Refer to Appendix A for error codes.

Parameter Discussion

The `testData` and `testError` structures can be ignored by pre and post-tests. They may optionally be used to share data between the pre-test, test, and post-test functions.

TX_SaveSequence

`int` **status** = **TX_SaveSequence** (char ***fileName**, void ***specialSaveFunction**);

Purpose

This function saves the current sequence in the specified file.

Parameters

Input	**fileName**	string	The name (and drive/path) of the file where the test sequence is saved
	specialSaveFunction	function pointer	Points to a function for saving additional information. See *Parameter Discussion* for more information.

Return Value

status	integer	Refer to Appendix A for error codes.

Parameter Discussion

You can specify a function to save additional data beyond what the Test Executive Engine saves with the **specialSaveFunction** parameter. For example, the Test Executive normally uses this function to save the preconditions. With this function, you can add extensions without having to modify the engine itself. This special save function has the following prototype, where **fileHandle** is the handle of the open sequence file and **numTests** is the number of tests in the sequence.

`int` **SpecialSaveFunc**(`int` **fileHandle**, `int` **numTests**)

TX_SetCurrTest

`int` **status** = **TX_SetCurrTest** (`int` **testNumber**);

Purpose

This function allows the Current Test Pointer to be moved to any test in the test sequence. The index of this function begins at one (1).

Parameters

Input	**testNumber**	integer	Index of the test where the CTP will point

Return Value

status	integer	Refer to Appendix A for error codes.

TX_SetEngineInfo

`int` **status** = **TX_SetEngineInfo** (unsigned short **field**, void ***newValue**);

Purpose

Copies information into the Engine and/or into the test that the CTP points to.

Notice that this function copies the data that is sent to the engine and stores this copy inside the engine. After you call this function, you can modify the data element sent to the engine without affecting the internally stored data.

Parameters

Input	**field**	short integer	Specifies what information to set. Refer to Appendix A for valid fields.
	newValue	void	Pointer to new value to be copied. Caller responsible for freeing memory used by this value

Return Value

status	integer	Refer to Appendix A for error codes.

TX_StrDup

`char *`**duplicate** = **TX_StrDup** (`char *`**src**);

Purpose

This function creates a duplicate copy of the string pointed to by **src** and returns a pointer to the new copy. `TX_StrDup` obtains the memory for the new string by using the `malloc()` function and you can free the memory using the `free()` function.

Parameters

Input	**src**	string	String to duplicate

Return Value

duplicate	string	Pointer to new copy of string. Null if unsuccessful

---

TX_VerifyPrePostTest

`int` **status** = **TX_VerifyPrePostTest** (`int` **executionTime**, `int` **testType,** `int` *__moduleId__, `int` *__functionStatus__,);

Purpose

This function verifies that the specified Sequence pre or post-test is available in the system.

Note: ***This function also loads the external module which contains the pre-test or post-test.***

Parameters

Input	**executionTime**	integer	If non-zero, the test runs only when the sequence is loaded or unloaded. If zero, the test runs every time the sequence is executed.
	testType	integer	If non-zero, the test is a pre-test. If zero, the test is a post test.
Output	**moduleId**	integer	Module ID number or error code. See **LoadExternalModule** for error codes.
	functionStatus	integer	Status of function pointer. See **LoadExternalModule** for error codes.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_VerifySequence

`int` **status** = **TX_VerifySequence** (`int` *__moduleId__, `int` *__functionStatus__, `int` *__testFailingVerify__);

Purpose

This function verifies that the local pre-test, post-test, and the main test functions are available in the system for every test in the system.

The tests are verified in order. If any test fails the verification, the function halts and returns the number of the test that failed.

This function does not verify the sequence pre- and post-test functions. **TX_VerifyPrePostTest**() must be run separately.

Note: ***This function also loads the external module which contains the test functions.***

Parameters

Output	**moduleId**	integer	Module ID number or error code. See **LoadExternalModule** for error codes.
	functionStatus	integer	Status of function pointer. See **LoadExternalModule** for error codes.
	testFailingVerify	integer	Index of first test which failed verification

Return Value

status	integer	Refer to Appendix A for error codes.

TX_VerifyTest

`int` **status** = **TX_VerifyTest** (`int *`**moduleId**, `int *`**functionStatus**)

Purpose

This function verifies that the local pre- post-test, and main test functions are available in the system for the test pointed to by the Current Test Pointer.

Note: ***This function also loads the external module which contains the test functions.***

Parameters

Output	**moduleId**	integer	Module ID number or error code. See **LoadExternalModule** for error codes.
	functionStatus	integer	Status of function pointer. See **LoadExternalModule** for error codes.

Return Value

status	integer	Refer to Appendix A for error codes.

TX_WriteByte

`int` **status** = **TX_WriteByte**(`int` **fileHandle**, `char` **byteToWrite**);

Purpose

This function writes a byte to the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file to which the function writes the value
	byteToWrite	string	Byte value to write to the specified file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_WriteDouble

`int` **status** = **TX_WriteDouble**(`int` **fileHandle**, `double` **doubleToWrite**);

Purpose

This function writes a double to the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file to which the function writes the value
	doubleToWrite	double-precision	Double-precision value to write to the specified file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_WriteInt

`int` **status** = **TX_WriteInt**(`int` **fileHandle**, `int` **integerToWrite**);

Purpose

This function writes a double-precision integer to the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file in which the function writes the value
	integerToWrite	integer	Integer value to write to the specified file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_WriteShort

`int` **status** = **TX_WriteShort**(`int` **fileHandle**, `short` **shortToWrite**);

Purpose

This function writes a short integer to the specified open file.

Parameters

Input	**fileHandle**	integer	The handle of an open file to which the function writes the value
	shortToWrite	short integer	Short integer value to write to the specified file

Return Value

status	integer	Refer to Appendix A for error codes.

TX_WriteString

`int` **status** = **TX_WriteShort**(`int` **fileHandle**, `char *`**stringToWrite**`);`

Purpose

This function writes a string to the specified open file.

Parameters

	fileHandle	integer	The handle of an open file to which the function writes the value
	stringToWrite	char	String to write to the specified file

Return Value

status	integer	Refer to Appendix A for error codes.

Appendix A
Error Codes and Engine Constants

This appendix lists the error codes and other important constants used by the Test Executive engine. See `txconst.c` for more information.

Error Codes

The following error codes are returned by functions in the Test Executive engine.

Table A-1. Error Codes

Error Code	Meaning
`ERR_CLIPBOARD_EMPTY`	Attempt to paste a test when the clipboard is empty
`ERR_END_OF_LIST`	Attempt to move past the beginning or end of the sequence
`ERR_ILLEGAL_POINTER`	Unexpected NULL value for pointer
`ERR_INVALID_ENGINE_INFO`	Constant passed to `TX_GetEngineInfo` or `TX_SetEngineInfo` is invalid
`ERR_INVALID_TEST_MODULE`	Problem unloading external module
`ERR_LIMIT_SPEC_OUT_OF_RANGE`	The measurement comparison type for the test is invalid
`ERR_LIST_EMPTY`	The sequence contains no tests
`ERR_LIST_NOT_EMPTY`	The sequence contains tests when it is expected to be empty
`ERR_LOCAL_POSTTEST_NOT_VERIFIED`	Attempt to execute unverified local post-test
`ERR_LOCAL_PRETEST_NOT_VERIFIED`	Attempt to execute unverified local pre-test
`ERR_METAPOSTTEST_RUN_TIME`	Run-time failure in sequence unload post-test
`ERR_METAPRETEST_RUN_TIME`	Run-time failure in sequence load pre-test
`ERR_META_POSTTEST_FAIL`	Sequence unload post-test failed
`ERR_META_PRETEST_FAIL`	Sequence load pre-test failed
`ERR_NEWER_VERSION`	Sequence file version is greater than Test Executive Engine version

(continues)

Table A-1. Error Codes (Continued)

Error Code	**Meaning**
ERR_NOT_TX_FILE	File is not a valid sequence file
ERR_NO_PRE_POST_TEST	Attempt to set or get information for undefined local pre-test or post-test
ERR_OUT_OF_MEMORY	Memory allocation attempt failed
ERR_POSTTEST_FAILED_VERIFY	Unable to verify local post-test. Usually due to invalid file or function name
ERR_POSTTEST_RUN_TIME	Run-time error in local post-test
ERR_PRETEST_FAILED_VERIFY	Unable to verify local preest. Usually due to invalid file or function name
ERR_PRETEST_RUN_TIME	Run-time error in local pre-test
ERR_PRE_POST_TEST_EXISTS	Attempt to define sequence pre-test or post-test when one already exists
ERR_READING_FROM_FILE	Error reading from file. Usually due to unmatched modifications to `TX_SaveSequence` and `TX_LoadSequence`
ERR_SEQPOSTTEST_RUN_TIME	Run-time error in sequence post-test
ERR_SEQPRETEST_RUN_TIME	Run-time error in sequence pre-test
ERR_SEQUENCE_POSTTEST_FAIL	The sequence post-test failed
ERR_SEQUENCE_PRETEST_FAIL	The sequence pre-test failed
ERR_TEST_FAILED_VERIFY	Unable to verify test. Usually due to invalid file or function name
ERR_TEST_NOT_VERIFIED	Attempt to execute unverified test
ERR_TEST_OUT_OF_RANGE	Attempt to set invalid test number with `TX_SetCurrTest`
ERR_TEST_RUN_TIME	Run-time error in test
ERR_UNABLE_TO_OPEN_FILE	Unable to open file
ERR_WRITING_TO_FILE	Error writing to file
NO_ERROR	No error

Engine Information Constants

The following constants are used by `TX_SetEngineInfo` and `TX_GetEngineInfo` to choose the information to set or retrieve.

Table A-2. Engine Information Constants

Constant	Type	Meaning
`INFO_GENERAL_STRING`	char * [1]	The sequence general purpose expansion string
`INFO_GOTO_TARGET`	int *	The index in the sequence of the test which is the destination of the goto
`INFO_IS_GOTO`	short *	A flag indicating whether the test is a goto
`INFO_LOOP_COUNT`	int *	The number of times the current test has looped
`INFO_MAX_LOOPS`	int *	Maximum number of times a test should loop
`INFO_METAPOSTTEST_NAMES`	char * [4]	The names associated with the sequence unload post-test: file name, function name, test name, and test description
`INFO_METAPOSTTEST_PARAMETERS`	char * [1]	The parameter string for the sequence unload post-test
`INFO_METAPRETEST_NAMES`	char * [4]	The names associated with the sequence load pre-test: file name, function name, test name, and test description
`INFO_METAPRETEST_PARAMETERS`	char * [1]	The parameter string for the sequence load pre-test
`INFO_POSTTEST_PARAMETERS`	char * [1]	The parameter string for the current test's post-test
`INFO_PRETEST_PARAMETERS`	char * [1]	The parameter string for the current test's pre-test
`INFO_REPORT_FILE_NAME`	char * [1]	The default name of the report file
`INFO_RPT_FILE_LOCK`	short *	Whether the operator can change the name of the report file

(continues)

Table A-2. Engine Information Constants (Continued)

Constant	Type	Meaning
INFO_RPT_FILE_MODE	short *	The report file mode: 0 for overwrite, 1 for append
INFO_SEQPOSTTEST_NAMES	char * [4]	The names associated with the sequence post-test: file name, function name, test name, and test description
INFO_SEQPOSTTEST_PARAMETERS	char * [1]	The parameter string for sequence post-test
INFO_SEQPRETEST_NAMES	char * [4]	The names associated with the sequence pre-test: file name, function name, test name, and test description
INFO_SEQPRETEST_PARAMETERS	char * [1]	The parameter string for sequence pre-test
INFO_SEQ_DESCRIPTION	char * [1]	The sequence description string
INFO_SEQ_FILE_NAME	char * [1]	The file name of the sequence. Used to detect when the sequence has moved
INFO_SIZE_OF_SEQUENCE	int *	The number of tests in the sequence
INFO_TEST_DESCRIPTION	char * [1]	Description string for an individual test
INFO_TEST_FAIL_ACTION	short *	The action to perform if the current test fails: LOOP_ACTION_CONT, LOOP_ACTION_LOOP, or LOOP_ACTION_STOP
INFO_TEST_LIMIT_HIGH	double *	The upper value to compare the test measurement against
INFO_TEST_LIMIT_LOW	double *	The lower value to compare the test measurement against
INFO_TEST_NAMES	char * [8]	The names associated with an individual test: file name, function name, pre-test file name, pre-test function name. post-test file name, post-test function name, test name, and test description

(continues)

Table A-2. Engine Information Constants (Continued)

Constant	**Type**	**Meaning**
INFO_TEST_NUMBER	int *	Index of the current test
INFO_TEST_PARAMETERS	char * [1]	Parameter string for the current test
INFO_TEST_PASS_ACTION	short *	The action to perform if the current test passes: LOOP_ACTION_CONT, LOOP_ACTION_LOOP, or LOOP_ACTION_STOP
INFO_TEST_RUN_MODE	short *	Run Mode for the test: RUN_MODE_NORMAL, RUN_MODE_FORCE_PASS, RUN_MODE_FORCE_FAIL, or RUN_MODE_SKIP
INFO_TEST_SPEC	short *	Type of comparison to make against measurement returned by test: SPEC_EQ, SPEC_NEQ, SPEC_GT, SPEC_GE, SPEC_LT, SPEC_LE, SPEC_GTLT, SPEC_GELE, SPEC_GTLE, SPEC_GELT, SPEC_BOOL, SPEC_LOG, or SPEC_NONE

Appendix B
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Corporate Headquarters
(512) 795-8248
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices	**Phone Number**	**Fax Number**
Australia	(03) 879 9422	(03) 879 9179
Austria	(0662) 435986	(0662) 437010-19
Belgium	02/757.00.20	02/757.03.11
Denmark	45 76 26 00	45 76 71 11
Finland	(90) 527 2321	(90) 502 2930
France	(1) 48 14 24 00	(1) 48 14 24 14
Germany	089/741 31 30	089/714 60 35
Italy	02/48301892	02/48301915
Japan	(03) 3788-1921	(03) 3788-1923
Mexico	95 800 010 0793	95 800 010 0793
Netherlands	03480-33466	03480-30673
Norway	32-848400	32-848600
Singapore	2265886	2265887
Spain	(91) 640 0085	(91) 640 0533
Sweden	08-730 49 70	08-730 43 70
Switzerland	056/20 51 51	056/20 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	0635 523545	0635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___)_____

Computer brand _____ Model _____ Processor _____

Operating system _____

Speed _____MHz RAM _____MB Display adapter _____

Mouse _____yes _____no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is _____

_____

_____

_____

_____

List any error messages _____

_____

_____

_____

_____

The following steps will reproduce the problem _____

_____

_____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **LabWindows®/CVI Test Executive Toolkit Reference Manual**

Edition Date: **November 1994**

Part Number: **320863A-01**

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678

Index

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W