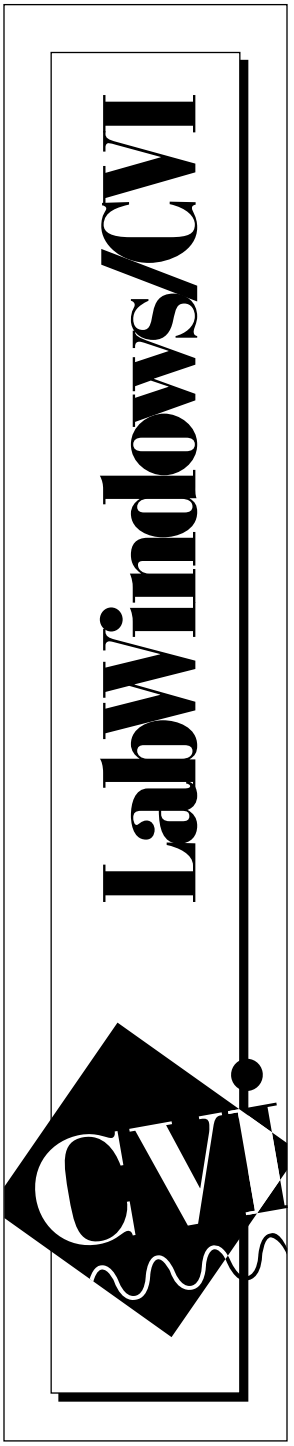


LabWindows®/CVI™ Test Executive Toolkit Reference Manual

July 1997 Edition
Part Number 320863B-01

Internet Support

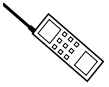
E-mail: support@natinst.com
info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

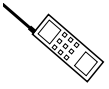
Fax-on-Demand Support

(512) 418-1111



Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678



International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30,
Hong Kong 2645 3186, Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51,
Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

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

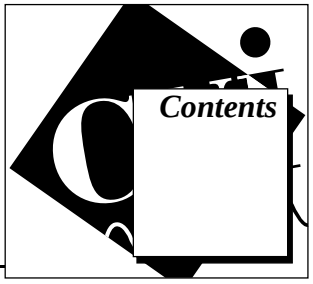
Trademarks

CVI™, natinst.com™, and National Instruments® are trademarks of National Instruments Corporation.

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.



About This Manual

Chapter 1
Introduction

Chapter 2
Getting Started

Chapter 3
Operating the Test Executive

Chapter 4
Creating Tests and Test Sequences

Chapter 5
Modifying the Test Executive

Chapter 6
Distributing the Test Executive

Chapter 7
Function Descriptions for the Test Executive Engine

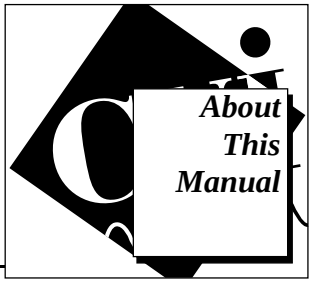
Appendix A
Error Codes and Attribute Constants

Appendix B
Customer Communication

Index

Figures

Tables



The *LabWindows/CVI Test Executive Toolkit Reference Manual* describes the features, functions, and operation of the LabWindows/CVI Test Executive Toolkit.

Organization of This Manual

The *LabWindows/CVI Test Executive Toolkit Reference Manual* includes the following sections:

- Chapter 1, *Introduction*, describes installation and lists the main features of the Test Executive Toolkit. It also explains the execution model and the three operating levels of the toolkit.
- Chapter 2, *Getting Started*, introduces the basic concepts of test executive operation and test sequence development.
- Chapter 3, *Operating the Test Executive*, describes 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 test functions and tests, and then combining them into test sequences. The last part of this chapter describes the controls and indicators of the Sequence Editor in which you create and edit test sequences.
- Chapter 5, *Modifying the Test Executive*, describes the organization and internal structure of the Test Executive and suggests where you can make modifications to the behavior of the Test Executive.
- Chapter 6, *Distributing the Test Executive*, describes the files required when you distribute a Test Executive executable.
- Chapter 7, *Function Descriptions for the Test Executive Engine*, describes the functions in the LabWindows/CVI Test Executive engine.
- Appendix A, Error Codes and Attribute Constants, lists the error codes and other important constants the Test Executive engine uses.

- Appendix B, *Customer Communication*, 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.
- 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 appear in this manual:

-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Ctrl-Alt-Delete>.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options»Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** option from the last dialog box.
bold	Bold text denotes the names of menus, menu items, parameters, dialog box buttons or options, icons, Windows 95 tabs, or LEDs.
bold italic	Bold italic text denotes an activity objective, note, caution, or warning.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, filenames, and extensions, and for statements and comments taken from program code.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in `C:\dir1name\dir2name\filename`.

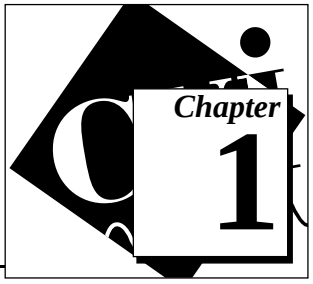
Related Documentation

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

- *Getting Started with LabWindows/CVI*
- *LabWindows/CVI User Manual*
- *LabWindows/CVI SQL Toolkit Reference 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 configuration forms for you to complete. These forms are in Appendix B, *Customer Communication*, at the end of this manual.



Introduction

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

Installation

The following section tells you how to install the Test Executive Toolkit on the Windows and UNIX platforms.

The LabWindows/CVI Test Executive comes in compressed form on floppy disk. Installing the Test Executive requires approximately 5 MB of space on your hard drive.

Windows 3.*x* and Windows NT 3.*x*

Insert disk one of the Test Executive Toolkit into the 3.5-inch disk drive and run the `setup.exe` program using one of the following methods:

- Under Windows, launch the File Manager. Click on the icon of the drive that contains the installation disk. Find `setup.exe` in the list of files on that disk and double-click on it to begin installation.
- Under Windows, choose **Run...** from the **File** menu of the Program Manager. In the dialog box that appears, type *X*`:\setup` (where *X* is the name of the drive that contains the installation disk). Click on **OK** to begin installation.

After you begin installation, follow the instructions that appear on the screen.

Windows 95 and NT

Insert disk one Test Executive Toolkit into the 3.5-inch disk drive and run the `setup.exe` program using the following methods:

- Under Windows 95 or NT, launch the Windows Explorer. Click on the icon of the drive that contains the installation disk. Find `setup.exe` in the list of files on that disk and double-click on it to begin installation.

- Under Windows 95, choose **Run...** from the **Start** pop-up menu. In the dialog box that appears, type *X*:\setup (where *X* is the name of the drive that contains the installation disk). Click on **OK** to begin installation.

After you begin installation, follow the instructions that appear on the screen.

UNIX

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

1. Change to the directory where you want to install the testexec directory structure.
2. Insert the Test Executive disk into the 3.5-inch disk drive.
3. Type the following UNIX command for the type of installation disk you chose:

   Solaris 1: `tar xvf /dev/fd0`

   Solaris 2: `tar xvf /dev/fd0`

   HP-UX: `tar xvf /dev/rfloppy/`*`devicename`*
   (where *devicename* is the floppy device as described in HP-UX documentation)

4. To run the installation script issue the following command:

   ```
   INSTALL
   ```

After you begin installation, follow the instructions that appear on the screen.

Updating Sequence Paths

When you install the Test Executive Toolkit on a target computer, you must change the paths in the example sequences to reflect their new directory. When opening an example sequence file the Test Executive detects when a sequence has moved and asks if you want to update the paths. When you click on **Yes**, the program automatically updates the paths and prompts you to save the changes. Click on **Yes** to save changes and facilitate the running of example sequences.

Product Overview

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

The Test Executive runs test programs using C functions and implements the following features:

- Test sequencing based on pass/fail status, preconditions, and goto commands
- Subsequencing
- Logging of test results to ASCII file or database (requires LabWindows/CVI SQL Toolkit)
- Run-time interfacing, including prompts for operator and Unit Under Test (UUT) 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
- Three operating levels (operator, technician, developer)
- Breakpoints on tests within sequences, and single-step debugging

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 the Test Executive does not prompt you 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.

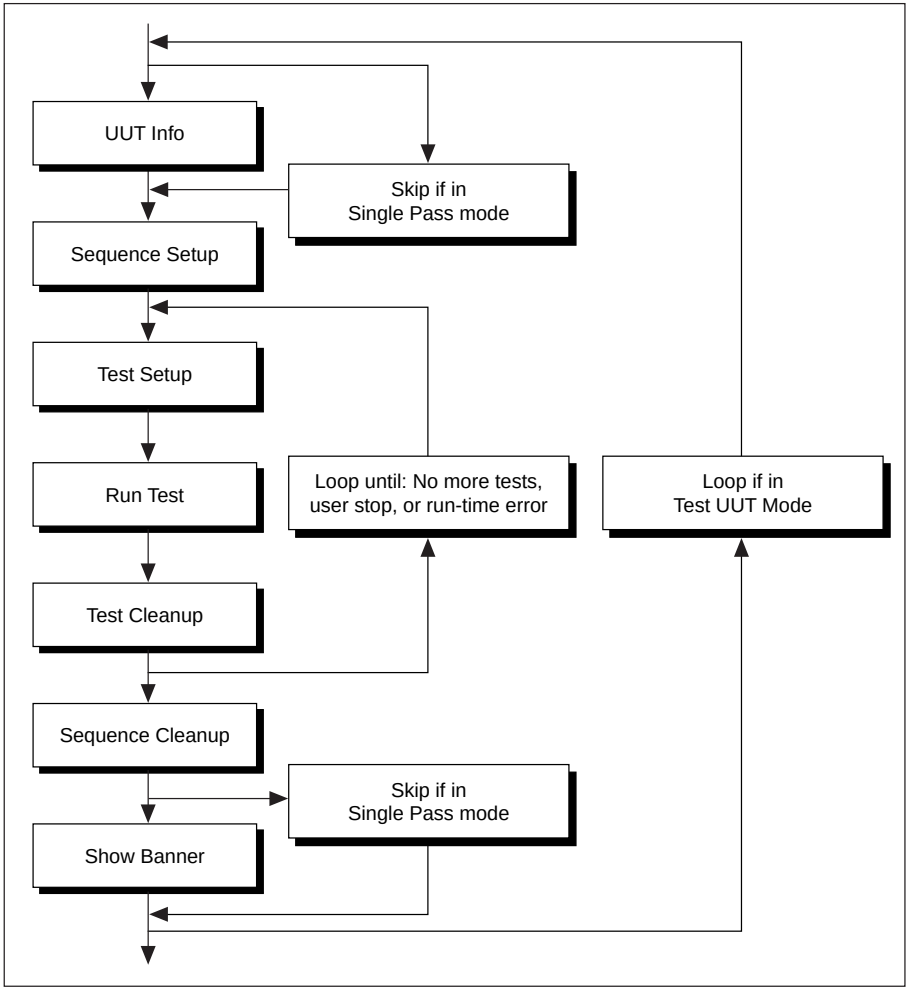


Figure 1-1. Flowchart for UUT Test and Single Pass Operation

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

Operating Levels

The Test Executive has three operating levels (Operator, Technician, and Developer). The following table summarizes the capabilities of each operating level.

Table 1-1. Text Executive Operating Levels

Level	UUT Test	Debug Sequences/Tests	Edit Sequences
Developer	Yes	Yes	Yes
Technician	Yes	Yes	No
Operator	Yes	No	No

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

At the Technician level you can execute tests to diagnose a UUT. You can run individual sequences and tests, but you cannot edit sequences. You can also set breakpoints and step through sequences.

At the Operator level you only can execute test sequences in UUT Test mode by clicking on the **Test UUT** button.

You set the operating level in the Login dialog box, which you access through the **File** menu. You can change the operator level at any time while the Test Executive is running. The operator level and default passwords appear in Table 1-2. The *Changing Passwords* section of Chapter 5, *Modifying the Test Executive*, tells how you can modify the default passwords.

Table 1-2. Default Login Names and Passwords

Operator Level	Password
developer	developer
technician	technician
operator	operator

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`. The `tstsuite.prj` project file adds database capabilities to the Test Executive when used with the LabWindows/CVI SQL Toolkit. You may want to save copies of these project files under different names for each of your development efforts.

The `testexec.prj` file contains all the source code files for the Test Executive. You can build a standalone executable from the `testexec.prj` file. You can also include your test functions as source code files in the `testexec.prj` file. However, programmers typically 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 Test Executive standalone executable.

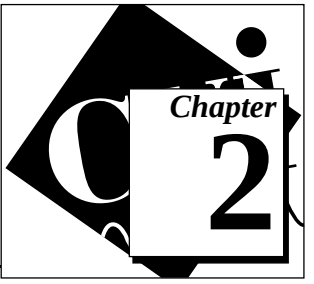
Use the following procedure as you develop test functions for the LabWindows/CVI Test Executive.

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 `\examples\template.c` contains example test 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. The standard Test Executive automatically uses the source file instead of the object file when you include the source file with the project.
3. Compile the source code into a module using LabWindows/CVI.
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 also specify the names of the test functions and the files in which they exist.

If you want to customize the LabWindows/CVI Test Executive, you can modify the Test Executive project source code with LabWindows/CVI. Refer to the project file which lists all of the user interface and source files for the Test Executive project.



Getting Started

This chapter introduces the basic concepts of test executive operation and test sequence development, using the examples described in the following table.

Table 2-1. Examples for Operation and Development of a Test Executive

Example	Relevant to Users Who...
Operating a test sequence	Operate the Test Executive
Examining a test function	Write test functions
Editing a test sequence	Create test sequences
Test sequence examples	Operate or develop a test executive

The examples in this chapter are sequential and each section depends on information from the previous section. The examples are designed to run in the Test Executive within the LabWindows/CVI environment. If you are executing the Test Executive as an standalone application, you can only work with the first example—*Operating 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 file, `testexec.prj`, in the `testexec` directory.
3. Choose **Run Project** from the **Run** menu to run the Test Executive. The login dialog box appears.
4. Enter your name and password as shown in Figure 2-1. The password you provide determines the operating level of the Test

Executive. For this example you work at the Operator level, so type `operator` in the Login Name and Password fields.

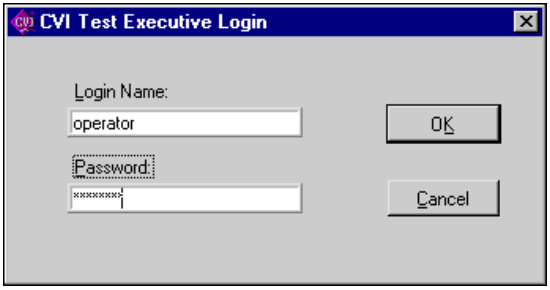


Figure 2-1. Login Dialog Box

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. The Changing Passwords section of Chapter 5, Modifying the Test Executive, tells how you can modify the default passwords.***

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 indicates status and the word Stopped indicates that no test sequence is currently running. Figure 2-2 shows the Test Executive front panel.

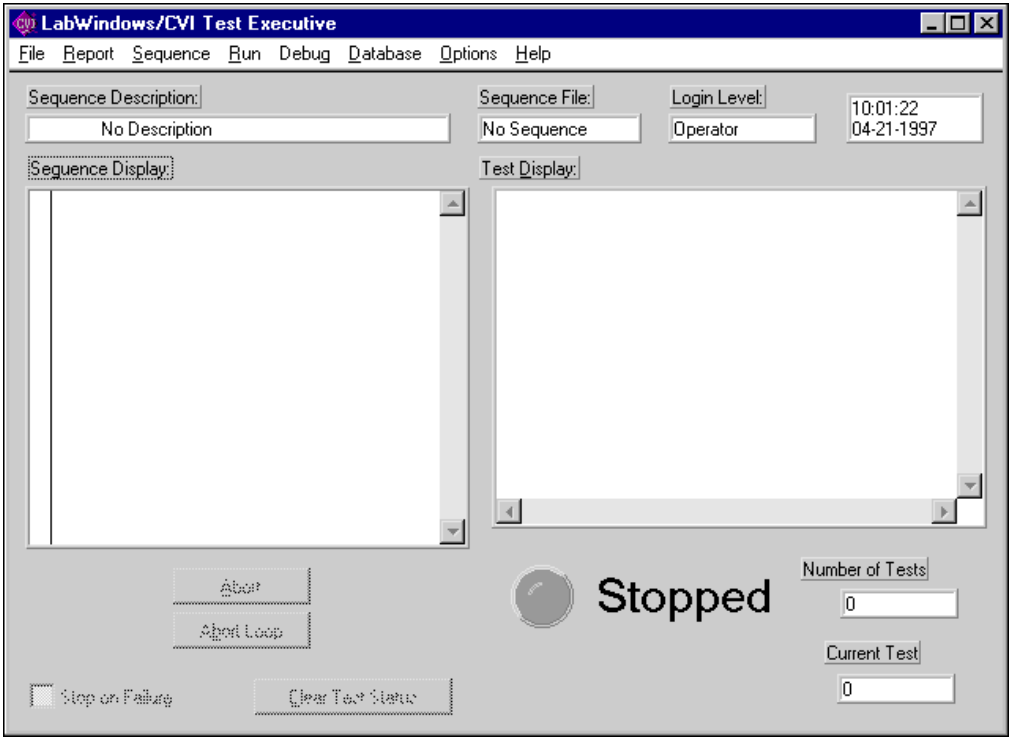


Figure 2-2. Test Executive Front Panel in Stopped Mode

Running a Test Sequence

Perform the following steps to run a test sequence.

1. Select **Open...** from the **File** menu of the Test Executive front panel.
2. Select the file `computer.squ` from the `examples` subdirectory of the `testexec` directory.

When you load `computer.squ`, notice that the steps of the test sequence appear in the Sequence Display. The name of the test sequence appears in the Sequence File indicator. The Computer test sequence list includes a folder, `cpu`, which contains a subsequence, `cpu.squ`.

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, you want to 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 the word Running and the LED flashes.
   - As each test runs, the word Running appears next to the name of the active test in the Sequence Display.
   - While the Test Executive executes the `cpu.squ` subsequence, the Sequence Display shows the tests in `cpu.squ` rather than the tests in `computer.squ`.
   - 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 or failed, or whether the user aborted the sequence.

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 prompt you for another UUT serial number 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 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 Tests**, and **Loop on**

Tests 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 Tests** button. Notice that only the selected test runs. Single Test execution runs a subset of tests that you select 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 on Tests** button. You can also use **Run Tests** to run multiple tests in the sequence. Place a checkmark beside the tests you want to run and click on **Run Tests**.

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 displays the updated Test Report.

Exiting The Test Executive

Select **Exit** from the **File** menu to dismiss the front panel 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 section 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 the structure of a test function.

1. Launch LabWindows/CVI if you have not already done so, but do not run the Test Executive project, yet.
2. Find and open the source file `random.c`, located in the `testexec\examples` directory.

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

```
void TX_FUNC RandomExample(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 = data->mallocFuncPtr(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 `RandomExample` as if it were a newly created test function to step through the test sequence creation process.

`RandomExample` 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 the measurement 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 see how these structure members are used.

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

You use this test function to create a test sequence, later in this chapter.

Creating and 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. Open the Test Executive project, `testexec.prj`, and run it. Enter `developer` in the Password field of the Login dialog box that appears 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 will show all the tests that you define for a test sequence. You use the Sequence Editor to input all of the specifications that define a test sequence.
4. Click on **New Test**. The test attributes controls appear below the test list box, as shown in Figure 2-3.

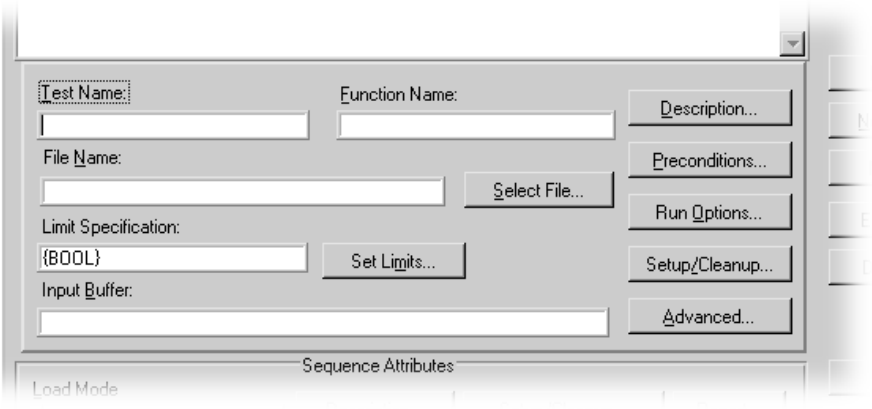


Figure 2-3. Test Attributes Child Panel

5. Click in the control labeled Test Name. In this example, you add the `RandomExample` function to a new sequence and configure its limit specifications. Type `Random-Boolean` in the Test Name control and `RandomExample` in the Function Name control.
6. 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`.

   You must give the Test Executive a limit specification that defines whether a test passes or fails. The Test Executive refers to the `tTestData` structure of the test function and applies the limit you

specify in the Sequence Editor to each test. The `tTestData` structure contains both a Boolean flag and a numeric measurement.

7. Click on the **Set Limits...** button to view the Set Limits for Test 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 for the comparison. For this example, set Comparison Type to `BOOL`. Your Set Limit Specification dialog box should match the settings in Figure 2-4.

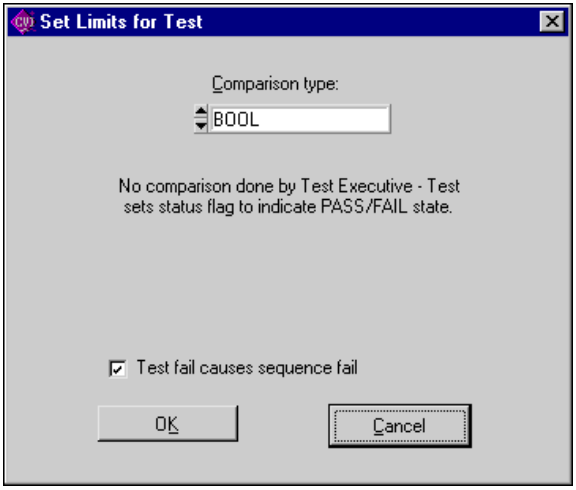


Figure 2-4. Set Limits for Test Dialog Box

When you set Comparison Type to `BOOL` the Test Executive uses the result flag in the `tTestData` structure to determine whether the test passed or failed.

8. Click on **OK**. The Limit Specification control now contains the text `{BOOL}`.
9. Next, add another test that uses a numeric limit specification, rather than a Boolean. Use the Test Name `Random-Numeric`. (Use the same test file function, `RandomExample`.)

10. Click on **Set Limits** 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 Figure 2-5.

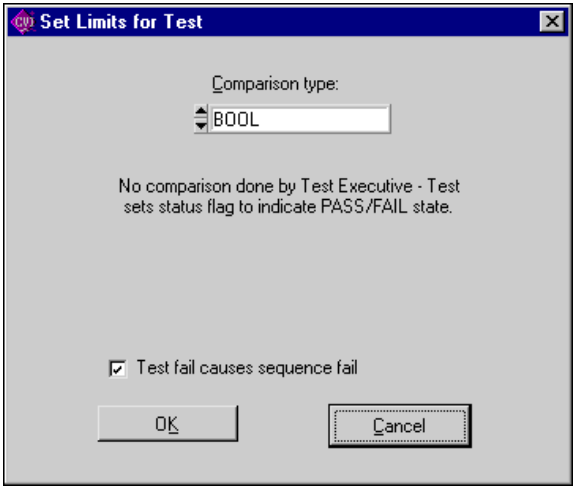


Figure 2-5. Setting Numeric Limits

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

12. Your completed test sequence should match the Sequence Editor list box shown in Figure 2-6.

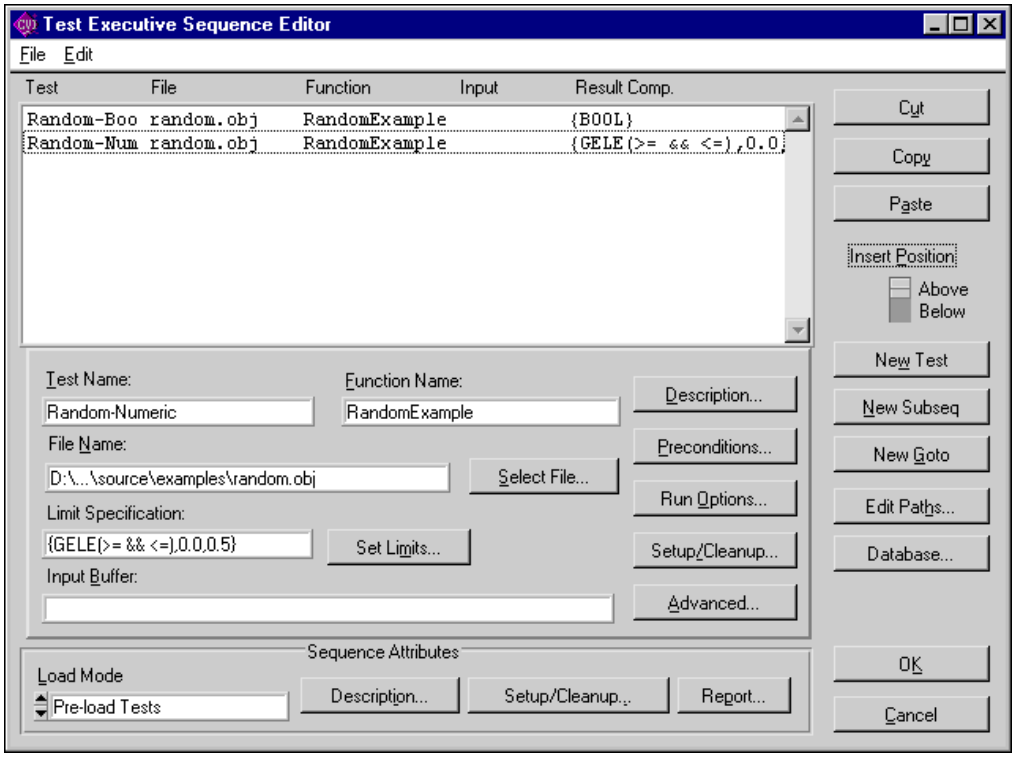


Figure 2-6. A Completed Test Sequence Setup

Before you save your new test sequence, you need to set preconditions and related attributes, as described in the following sections. Remember that you can edit any test in the Sequence Editor by clicking on the test you want to modify in the Sequence Editor list box. The specifications of the test appear in the Test Attributes controls located below the list box. As you make changes in the controls, the list box displays your modifications.

☞ **Note:** ***To enter attributes for a new test you must click on the* New Test *button.***

Setting Preconditions

In the following steps you set up a dependency between the two tests in the sequence you created in the previous section.

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

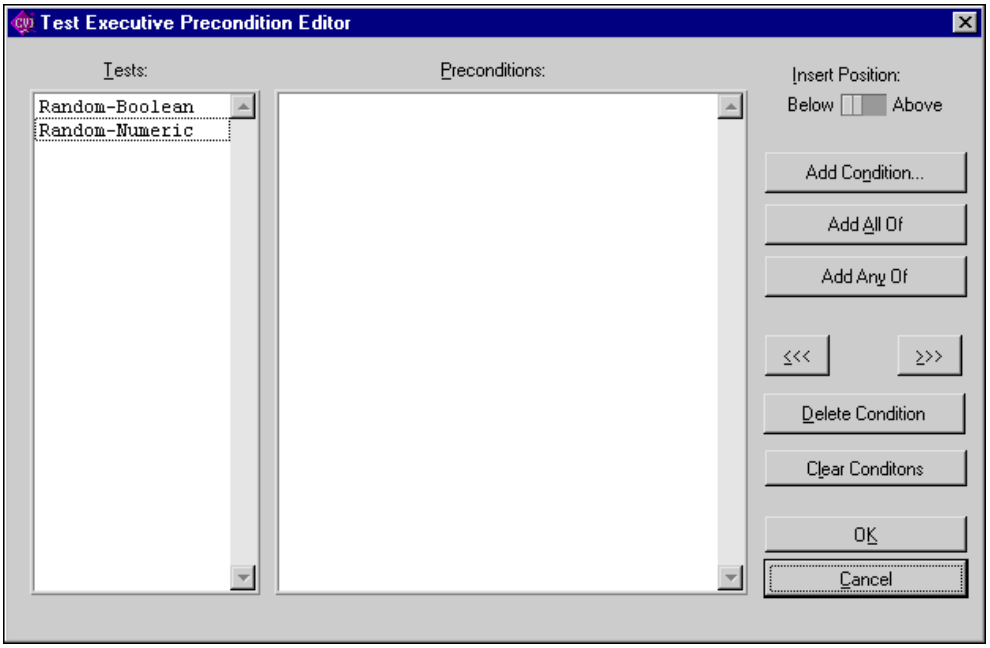


Figure 2-7. Precondition Editor

2. Click on `Random-Numeric` in the Tests list box. Then click on **Add Condition...**.
3. In the Add Condition dialog box that appears, set up the following dependency for `Random-Numeric`: that `Random-Boolean` must pass before `Random-Numeric` can execute.
   - Set the Type switch to `PASS` and click on `Random-Boolean` in the Tests list box. A checkmark appears beside the test name. Your settings should match those in Figure 2-8.
   - Click on **OK**.

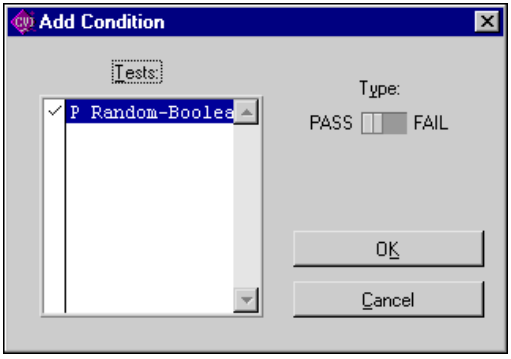


Figure 2-8. Using Add Condition to Set Preconditions

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

☞ **Note:** ***The Preconditions list box displays only the preconditions for the test you have selected in the Tests list box.***

4. Your Precondition Editor dialog box should look similar to the one in Figure 2-9. Click on **OK** to save the new dependency specification and return to the Sequence Editor.

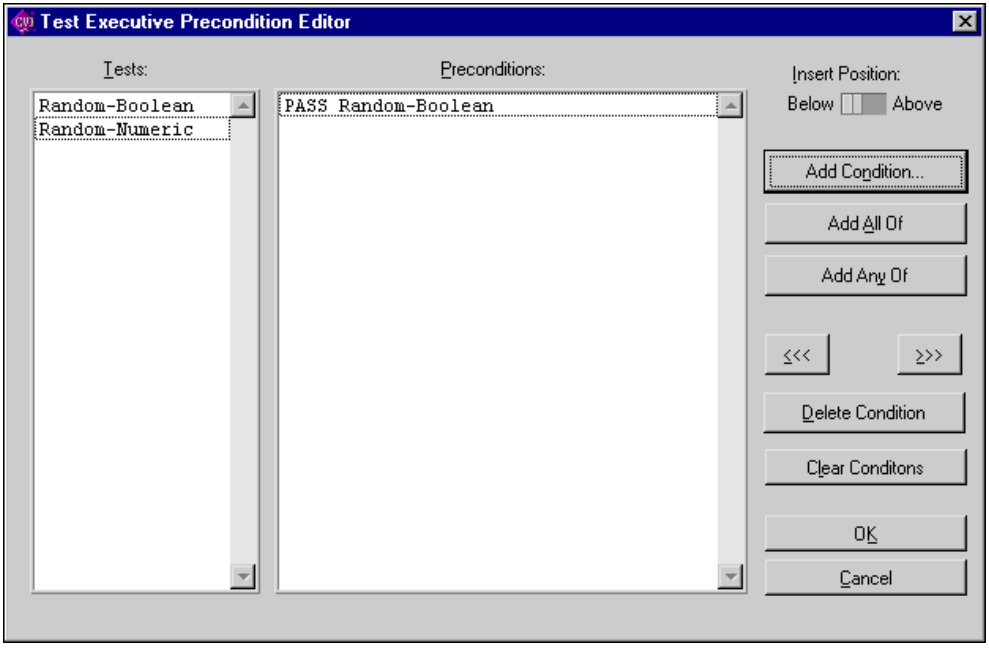


Figure 2-9. A Completed Random-Boolean Precondition Setup

Setting the Report File

You can configure the Test Executive to report test results by specifying a report file to receive those results. The following steps tell you how to specify the default report file.

1. Click on the **Report…** button in the Sequence Attributes section. The Set Default Report File dialog box appears.
2. Use the **Select File…** button to select `random.rpt` as the default report file.

3. Click in the **Lock File Name** box. Checking this box locks the filename to the value in the Test Report File control. If you do not lock the file name, the system gives you the opportunity to change the filename, each time you open the sequence file.

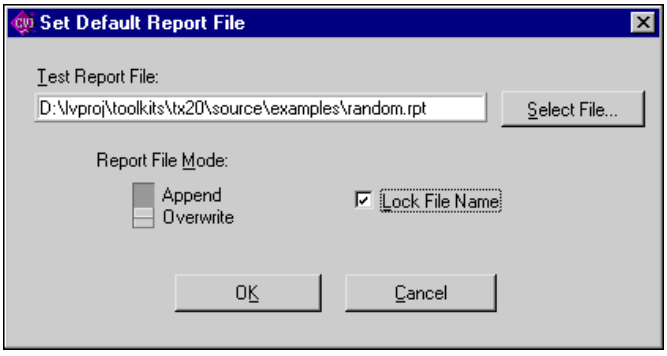


Figure 2-10. Specifying the Default Report File.

4. Click on the **OK** button to return to the Sequence Editor.
5. Click on **OK** in the Sequence Editor to return to the Test Executive front panel. Click on **No** when the program prompts you to save you sequence.

You are now ready to run your test sequence.

Running the Sequence

To run your test sequence, perform the following steps:

1. Run your new test sequence. The Test Executive will automatically determine whether 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. Click on **Test UUT**, enter a serial number of you choice, and click on **OK**.
   b. Perform tests for several UUT serial numbers and then click on the **Stop** button in the UUT Information dialog box.
   c. Select **View** from the **Report** menu to see the data that the `RandomExample` function generated for each test.
2. Choose **Exit** from the **File** menu to exit from the Test Executive. The application prompts you to save the sequence you created. To maintain this example in its original state, do not save your changes.

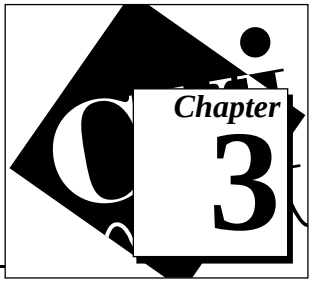
The three examples presented up to this point in the chapter show the fundamental operations in the Test Executive and also developer-level operations: creating test functions and using the Sequence Editor to develop test sequences that use these functions.

Example Sequences

The Test Executive package includes several test sequence examples located in the `examples` subdirectory of the Test Executive. The sequences demonstrate many aspects of the Test Executive.

- `auto.squ` simulates automobile subsystem testing. It includes setup and cleanup functions (functions that run before or after a test or test sequence) as well as load and unload functions (functions that run when the sequence is loaded or unloaded), multiple preconditions, and tests that use a variety of comparison types.
- `comment.squ` executes tests that use an **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` simulates testing a computer motherboard. It includes setup and cleanup functions, a subsequence (`cpu.squ`), multiple preconditions, and tests that use a variety of comparison types.
- `loopindx.squ` shows how a subsequence obtains the looping index of a test which is looping.
- `loopxmpl.squ` demonstrates how to set up loops over multiple tests.
- `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 how setup and cleanup functions work.

Operating the Test Executive



This chapter describes 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.

Figure 3-1 shows the Test Executive front panel at the Developer level.

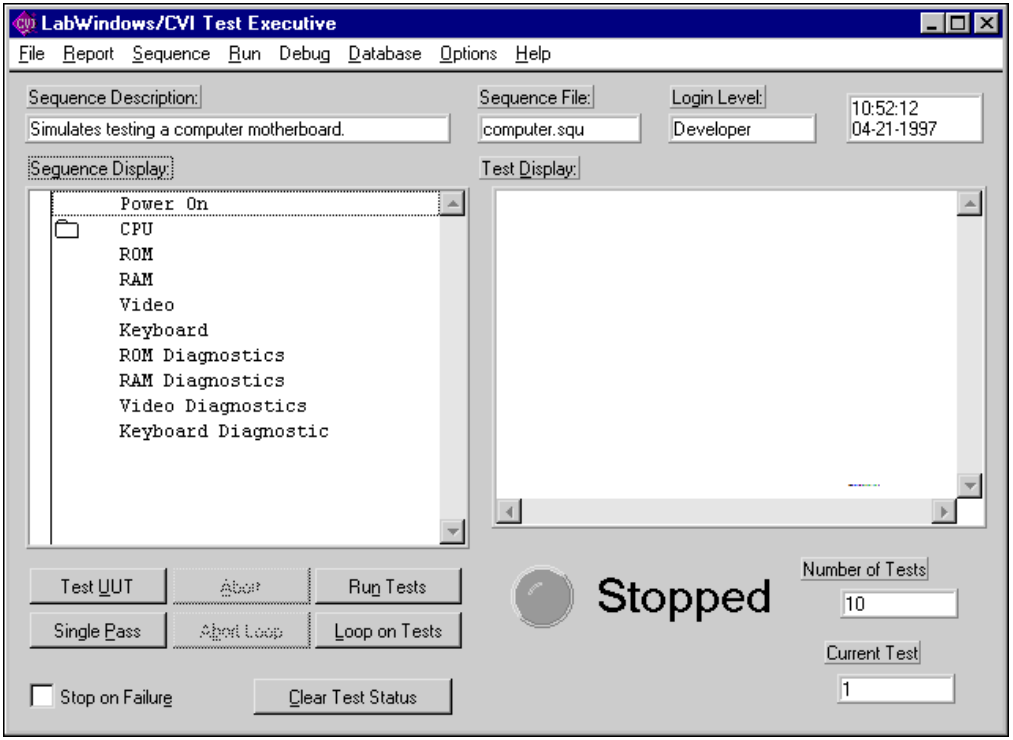


Figure 3-1. Test Executive Main Panel

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 the Test Executive controls.

File Menu

Login...

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

New

The **New** menu item creates a new, empty sequence. You can use the **New** menu item only at the Developer operating level.

Open...

The **Open...** menu item opens an existing sequence file. You can use the **Open...** menu item at any operating level.

Close

The **Close** menu item closes an open sequence file. You can use the **Close** menu item at any operating level.

Save

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

Save Copy As...

The **Save Copy As...** menu item saves a test sequence to file. A dialog box appears, prompting you to name the file. You can use the **Save Copy As...** menu item only at the Developer operating level.

Print...

The **Print...** menu item displays a dialog box where you select a report file to print. You can use the **Print** menu item at any operating level.

Exit

The **Exit** menu item causes the Test Executive to stop execution. If you have modified any of the currently loaded sequences, the Test Executive prompts you to save your changes before quitting. You can use the **Exit** menu item at any operating level.

Report Menu

View

The **View** menu item displays the current test report. You can view the report with the built-in report viewer or an external program such as Notepad. (The *Options Menu* section in this chapter describes your options for viewing reports). You can use the **View** menu item at any operating level.

The Test Report contains 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
```

Delete

The **Delete** menu item invokes a dialog box where you can remove from disk the test report (`.rpt`) file for the test sequence (`.squ`) file that is open on the Test Executive front panel. You can access the **Delete** menu item at any operating level.

Screen and File

The **Screen** and **File** menu items control which tests results display on screen or appear in the report file. You can choose to display or save all results, only pass results, only fail results, or no results. You can use the **Screen** or **File** menu items at any operating level.

Sequence Menu

Edit Sequence...

The **Edit Sequence...** menu item invokes the Sequence Editor. You can use the **Edit Sequence...** menu item only at the Developer operating level.

View Description...

The **View Description...** menu item invokes a dialog box that displays the entire sequence description. You can access the **View Description...** menu item at any operating level.

Generate Documentation...

The **Generate Documentation...** menu item generates a file that documents the settings for each test in the current sequence. You can use the **Generate Documentation...** menu item at any operating level.

Sequence Names

The names of all loaded sequences also appear in the **Sequence** menu. You can select a sequence name to make that sequence the active sequence in the Test Executive front panel.

Run Menu

Test UUT

The **Test UUT** menu item 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** menu item at any operating level.

Single Pass

The **Single Pass** menu item 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 use the **Single Pass** menu item only at the Developer and Technician operating level.

Run Tests

The **Run Tests** menu item executes the tests you select in the Sequence Display. You can select multiple tests by placing a checkmark beside each test. The **Run Tests** menu item is available only at the Developer and Technician operating levels.

Loop on Tests

The **Loop on Tests** menu item starts a repetitive execution of the tests currently selected in the Sequence Display list box. You can only access

the **Loop on Tests** menu item from the Developer and Technician levels. When you select **Loop on Tests**, the dialog box shown in Figure 3-2 appears.

Figure 3-2. Loop Test Dialog Box

Your loop test 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 test that will iterate five times. If you want looping to stop when the test fails, click on the Stop if test Fails checkbox.

Sequence Pre Test

The **Sequence Pre Test** menu item forces execution of the setup test for a sequence. The setup test for a sequence normally executes automatically before the sequence executes.

Sequence Post Test

The **Sequence Post Test** menu item forces execution of the cleanup test for a sequence. The cleanup test for a sequence normally executes automatically after the sequence executes.

Sequence Load Test

The **Sequence Load Test** menu item forces execution of the load test for a sequence. The load test for a sequence normally executes automatically when a sequence loads.

Sequence Unload Test

The **Sequence Unload Test** menu item forces execution of the unload test for a sequence. The unload test for a sequence normally executes automatically when a sequence unloads.

Debug Menu

☞ **Note:** ***The* Debug *menu also appears as a pop-up menu when you right-click on a test in the Sequence Display listbox.***

Toggle Breakpoint

The **Toggle Breakpoint** menu item sets or clears a breakpoint on the currently selected test in the Sequence Display. You can use the **Toggle Breakpoint** menu item only at the Developer or Technician operating level. You use the other commands in the **Debug** menu to move beyond the breakpoints you set using **Toggle Breakpoint**.

Break at First Test

The **Break at First Test** menu causes the Test Executive to break execution at the first test. You can use the **Break at First Test** menu item at any operating level.

Pause

The **Pause** menu item stops sequence execution and enters breakpoint mode. You can use the **Pause** menu item at any operating level.

Continue

The **Continue** menu item ends breakpoint mode and resumes execution. You can use the **Continue** menu item at any operating level.

Step Over

The **Step Over** menu item executes the current test in the sequence. Execution stops before the next test in the current sequence. You can use the **Step Over** menu item only at the Developer or Technician operating level.

Step Into

The **Step Into** menu item executes the current test in the current sequence. If the test is a subsequence, execution stops before the first test in the subsequence. Otherwise, execution stops before the next test in the current sequence. You can use the **Step Into** menu item only at the Developer or Technician operating level.

Set Next Test to Cursor

Set Next Test to Cursor sets the currently selected test in the Sequence Display as the next test to execute. You can use **Set Next Test to Cursor** to skip tests during debugging. The **Set Next Test to Cursor** menu item is available only at the Developer or Technician operating level.

Finish Sequence

The **Finish Sequence** menu item executes the remaining tests in the current sequence and reenters breakpoint mode before the next test in the parent sequence. You can use the **Finish Sequence** menu item only at the Developer or Technician operating level.

Terminate Execution

The **Terminate Execution** menu item exits breakpoint mode and terminates sequence execution without executing more tests. This command is equivalent to **Abort**. You can use the **Terminate Execution** menu item only at the Developer or Technician operating level.

Normal

The **Normal** menu item sets the run mode of the selected test to normal, instead of Force to Pass, Force to Fail, or Force to Skip. You can use the **Normal** menu item only at 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 use the **Force to Pass** menu item only at 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 at 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 the menu items* 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.***

Database Menu (tstsuite.prj only)

☞ **Note:** ***The* Database *Menu appears only when you run the Test Suite project file (*`tstsuite.prj`*).***

View Database...

The **View Database...** menu item invokes the Database Results Viewer. In the results viewer, you can see an overview of the sequence and test results. The results viewer also displays the details of any sequence or test result. You can access the **View Database...** menu item at any operating level.

Delete Database

The **Delete Database** menu item deletes the sequence and test result tables for the current sequence. You can access the **Delete Database...** menu item only at the developer operating level.

Logging Enabled

The **Logging Enabled** menu item activates logging to database tables. You can change **Logging Enabled** only at the developer and technician operating levels.

Options Menu

Report...

The **Report...** menu item invokes the Report Options dialog box. In the Report Options dialog box you can choose between viewing reports in the Test Display box, the built-in report viewer or an external report viewer. You can access the **Report...** menu item at any operating level.

Controls of the Front Panel

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 at 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 use the **Single Pass** button only at the Developer and Technician operating levels.

Abort

The **Abort** button stops execution of the test sequence after the current test completes execution. When you have clicked on **Test UUT** to start testing, clicking on **Abort** stops testing for the current UUT after the current executing test completes. The system then prompts you for the next UUT serial number. The **Abort** button is available at all operating levels, but is active only while a test is running.

Abort Loop

Clicking on the **Abort Loop** button stops the execution of a loop that is defined for a single test. (You define a loop for a single test in the Run Options dialog box of the Sequence Editor.) 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 use the **Abort Loop** button at any operating level.

Run Tests

The **Run Tests** button executes the tests that you select in the Sequence Display. You can select multiple tests for execution by placing a checkmark beside each test. The **Run Tests** button is available only at the Developer and Technician operating levels.

Loop on Tests

The **Loop on Tests** button starts a repetitive execution of the test or tests that you select in the Sequence Display list box. You can only access the **Loop on Tests** button from the Developer and Technician levels. When you select **Loop on Tests**, the dialog box shown in Figure 3-3 appears.

Figure 3-3. Loop Test Dialog Box

You can set the number of times the test(s) execute and also set the loop to stop if the test fails. To set the number of iterations, enter the number of iterations you want in the Loop control. Figure 3-3 shows a loop that will iterate five times. If you want looping to stop when the test fails, place a checkmark in the Stop if test Fails checkbox.

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 use the **Stop if test FAILs** box at 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 immediately when 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 that appear on the Test Executive front panel.

Sequence Description

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

Sequence File

The Sequence File indicator located above the Test Display shows the name of the test sequence file that you have loaded.

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 shown in Figure 3-4 displays the list of tests for the test sequence you have loaded.

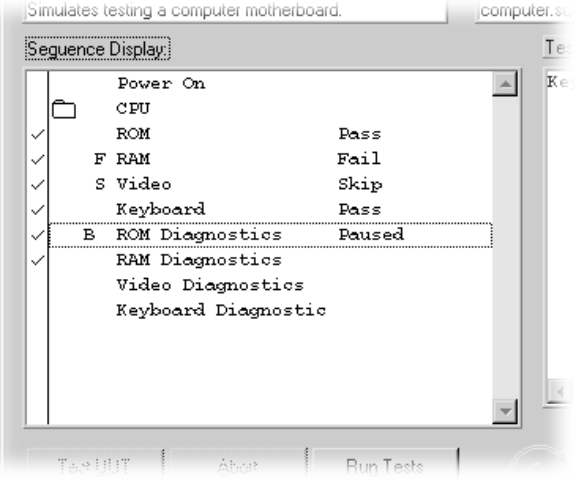


Figure 3-4. Sequence Display List Box

You see a checkmark column on the left side of the sequence display. The checkmarks you place in the column determine which tests run when you click on **Run Tests** or **Loop on Tests**. Checkmarks do not affect the action of the **Test UUT** or **Single Pass** buttons.

Each display line, such as the ones shown in Figure 3-4, contains several columns, with names, status codes, and icon to tell you characteristics of each test. From left to right, the columns are the checkmark field, and the indicators for subsequence, breakpoint, run mode, Test Name, and Test Status/Result.

The subsequence indicator displays a folder icon for subsequences.

The breakpoint indicator is blank when there is no breakpoint on the test or `B` when a breakpoint is present.

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)	The test runs normally.
`S`	The test is skipped.
`P`	Test is skipped with a forced `PASS` result.
`F`	Test is skipped with a forced `FAIL` result.

The Test Name field shows the name of the test.

The Test Status/Result field displays the word Running during test execution to indicate the active test. After the test completes, the field

displays 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.
`Running`	The Test Executive is running the test.
`Error`	Run-time error occurred during test execution.

Test Display

The Test Display shows the result of each test as it executes as well as certain error messages. Figure 3-5 shows the Test Display.

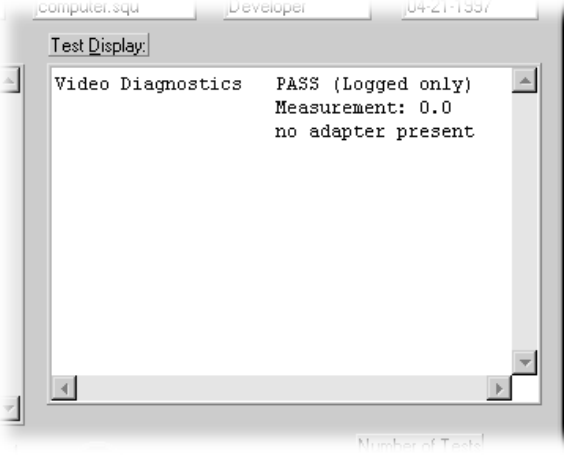


Figure 3-5. Test Display

Result of Each Test

After a test executes, the Test Display shows the complete result of that test. A test result takes the following form.

```
Test Name Result
Comment     (Comment may have multiple lines and is an optional field)
            Measurement Comparison Lower Limit Upper Limit
```

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

The specific test that logs a comment determines format and content of the `Comment` line(s). The `Measurement` field shows the measured value that the test returned. `Comparison` shows the type of limit checking that determined whether the test passed. The Condition for Test to Pass column in the following table shows the lower and upper limit values that determine whether a test passes or fails. The possible values of `Comparison` and their relation to the `Lower Limit` and `Upper Limit` (Condition for Test to Pass) appear in Table 3-3.

Table 3-3. Comparison Type Values

Comparison Type	Value	Condition for Test to Pass
Equal to	`EQ`	Measurement = Lower Limit
Not equal to	`NE`	Measurement != Lower Limit
Greater than	`GT`	Measurement > Lower Limit
Less than	`LT`	Measurement < Upper Limit
Greater than or equal to	`GE`	Measurement >= Lower Limit
Less than or equal to	`LE`	Measurement <= Upper Limit
Greater than—Less than	`GTLT`	Measurement > Lower Limit and < Upper Limit
Greater than or equal to—Less than or equal to	`GELE`	Measurement >= Lower Limit and <= Upper Limit

Table 3-3. Comparison Type Values (Continued)

Comparison Type	Value	Condition for Test to Pass
Greater than or equal to—Less than	GELT	Measurement >= Lower Limit and < Upper Limit
Greater than—Less than or equal to	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.

Status

The Status indicator, directly below the Test Display, shows the current operating status of the Test Executive. An LED calls attention to the Status indicator during a testing session. The possible values of the Status indicator and their meanings appear in Table 3-4.

Table 3-4. Status Indicator Values

Value	Meaning
Stopped	No test sequence is currently running.
Running	Test sequence is running.
Paused	Execution is paused at a breakpoint.
Looping	Test sequence is looping on a test or set of tests.

Operator Dialog Boxes

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

Login

The login dialog box prompts the operator for Login Name and Password. Figure 3-6 shows the login dialog box.

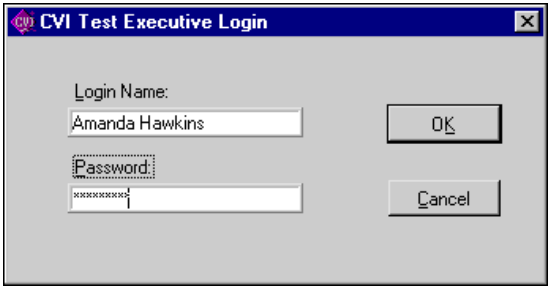


Figure 3-6. Login Dialog Box

Type the appropriate Password to set the desired operating level. Table 1-2 in Chapter 1, *Introduction*, lists the default passwords. The login 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 you make in the Name and Password controls. 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 starts at Operator level.

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.

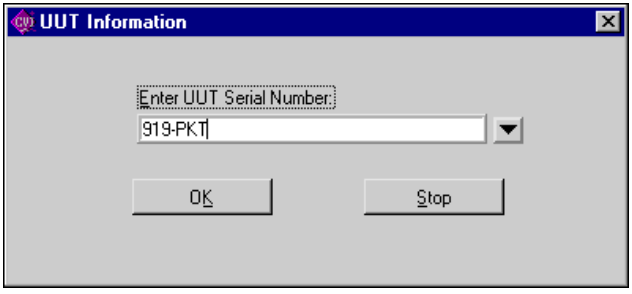


Figure 3-7. UUT Information Dialog Box

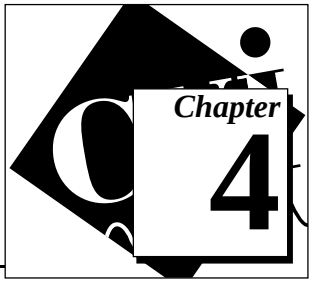
The UUT Information dialog box appears only when you use the **Test UUT** button. You can assign any ASCII string as the serial number for a test. That ASCII string will appear in the report for that test. You can click on the triangle button shown in Figure 3-7 to review or select previous serial numbers. 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 indicate whether the current UUT passed or failed. The Pass, Fail, or Abort banner appears at the end of test sequence execution for each UUT tested. These banners appear only when you are using the **Test UUT** button. Press <Enter> or click **OK** to acknowledge the banner and continue testing.

Figure 3-8. Pass Banner for Test Sequences

Creating Tests and Test Sequences



The first part of this chapter describes the process of creating test functions and tests, and then combining them into test sequences. The last part of this chapter describes the controls and indicators of the Sequence Editor where you create and edit test sequences.

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:

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

When you write functions, you may find the following process to be most efficient:

1. Create and debug your test functions in LabWindows/CVI, being sure to use the fixed prototype.
2. Include your functions in your source code files.
3. Add your source code files to the project file `testexec.prj`, in order to take advantage of the project debugging features of LabWindows/CVI.
4. After debugging the project, use LabWindows/CVI to create object modules from your source code files and remove the source code files from your project file.

Test Data Structure

A test function uses the Test Data structure to transmit data results that the Test Executive uses to determine whether a test passed or failed. The Test Data structure is defined as follows:

```
typedef struct TestData {
     Status           result;
     double           measurement;
     char             *inBuffer;
     char             *outBuffer;
```

```
        char        const  *modPath;
        char        const  *modFile;
        void               *hook;
        int                hookSize;
        tMallocPtr  const  *mallocFuncPtr;
        tFreePtr    const  *freeFuncPtr
} tTestData;
```

Table 4-1 describes the elements of the Test Data structure.

Table 4-1. Elements of the Test Data Structure

Name	Type	Meaning
`result` (Pass/Fail Flag)	`Status` (integer)	Value returned to indicate whether test passed or failed (This flag observed only if the sequence test step is set to pass or fail based on a Boolean comparison) `PASS(1)`; `FAIL(0)`
`measurement`	double-precision	Measurement value returned to evaluate Pass/Fail
`inBuffer`	string	String passed by Test Executive
`outBuffer`	string	String returned by test function (optional)
`modPath`	string	Directory path of file that contains the test function
`modFile`	string	File name of file containing test function
`hook`	generic pointer	Additional user-defined data
`hookSize`	integer	Size of the user-defined data
`mallocFuncPtr`	`tMallocPtr`	Pointer to the Test Executive `malloc` function
`freeFuncPtr`	`tFreePtr`	Pointer to the Test Executive `free` function

The Test Executive allocates and frees an *input buffer*, `inBuffer`, when one is specified for the test in the sequence. The test function can optionally return a string value, `outBuffer`, but the test must allocate the buffer, and the Test Executive will free 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 filename. These fields help you avoid problems if you later move the module that contains the test. The hook parameter gives you a way to pass arbitrary data to the test function. The example Test Executive ignores the hook parameter. The mallocFuncPtr and freeFuncPtr parameters give you a way to ensure that the Test Executive and the test functions use the same functions to allocate and to free memory. This consistency is especially important when you compile your tests as DLLs with one compiler and use a different compiler to compile the Test Executive.

To allocate data, use the following syntax:

```
myptr=data->mallocFuncPtr(NumberOfBytes);
                    /* mytptr=malloc(NumberOfBytes); */
```

To free data, use the following syntax:

```
data->freeFuncPtr(Pointer);         /* free(Pointer);*/
```

Test Error Structure

A test function uses the tTestError structure to report run-time errors. The error structure is defined as follows:

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

Table 4-2 describes elements of the error structure.

Table 4-2. Elements of the Error Structure

Name	Type	Meaning
errorFlag	Boolean (integer)	True(1) if an error occurred; False(0) otherwise
errorLocation	tErrLoc	Reserved for Test Executive internal use; where error occurred (in the test function itself or in 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 whether a run-time error occurred and then takes appropriate action.

Creating Setup and Cleanup Functions

Setup and cleanup functions—also called the pre and post 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 identifies the condition as a run-time error. Setup and cleanup functions do not log data; they only return status and run-time error information.

The prototype for the setup and cleanup function is defined as follows:

```
int TX_FUNC SampleSetup (char *inBuffer,
                         tTestData *data,
                         tTestError *error);
```

A return value of `1` means success; a return value of `0` is a run-time error.

Sample Test Templates

The `examples` subdirectory contains a collection of sample test templates to help you develop tests. Each test example shows how to use particular features of the Test Executive, including methods for development of both Boolean and limit tests, using subsequences, assigning error codes and messages to test operations, passing messages back to the Test Executive, and using buffer information passed into a test. When you develop new tests, start with one of the test templates in the `examples` directory to ensure smooth integration into the Test Executive.

Using the Sequence Editor

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 where it exists.

To create or modify a test sequence using the Sequence Editor, you must log in at the Developer level. The *Operating Levels* section of Chapter 1, *Introduction,* describes the login levels. Next you select **Edit Sequence...** from the **Sequence** menu. Figure 4-1 shows the Sequence Editor dialog box.

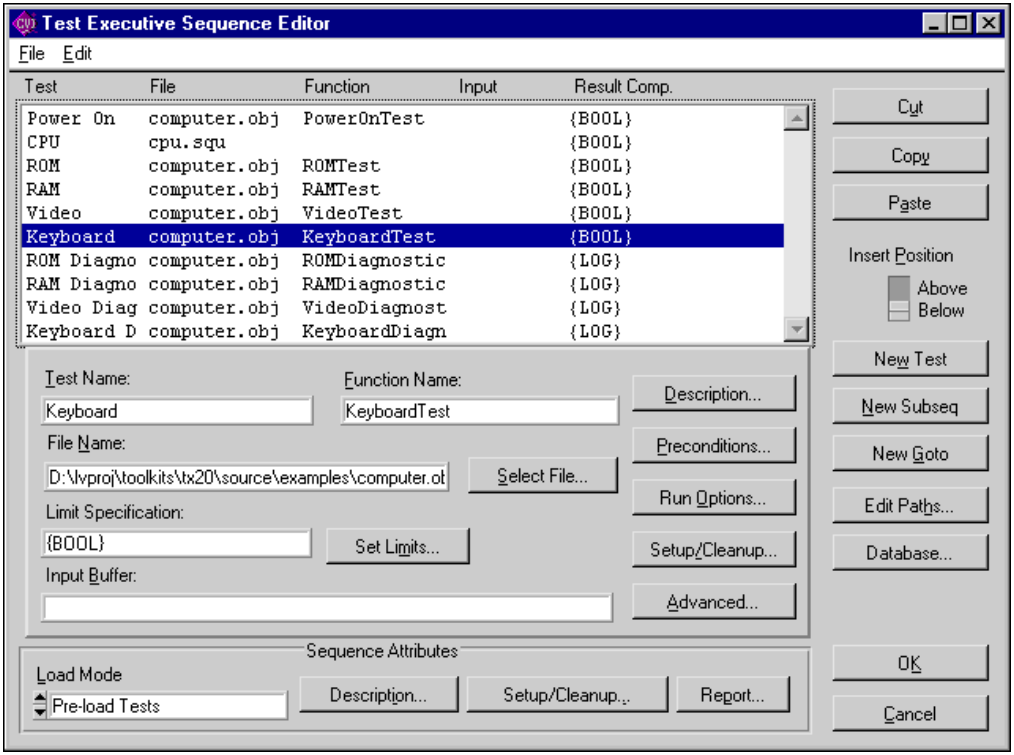


Figure 4-1. Sequence Editor Dialog Box

Test Sequence Overview

Keep the following points in mind as you create test sequences using the Sequence Editor. 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 or subsequence to perform the required testing operation. A typical test sequence has a list of tests, setup/cleanup functions, preconditions for flow control based on Pass/Fail results, test report file information, description of the sequence, and database information.

The tests that comprise a test sequence contain a combination of specifications that tell the Test Executive how to perform a single execution step in the testing process. A test specifies the following types of information:

- Name of the test as it appears in the Test list of the Sequence Editor
- Function or subsequence to execute
- Input Buffer that contains data for test function
- Limit Specification to define when a test passes or fails
- Run options specifying how a test will execute
- Fail and Pass actions (and maximum loop count, if applicable)
- The load mode of a test
- Database and report behavior

Controls for Editing Tests

When you edit a new or existing test in the Sequence Editor, the test attributes area appears directly below the Test list box, as shown in Figure 4-2. These attribute controls include Test Name, Function Name, File Name, Select File, Limit Specification, Set Limits, Input Buffer, Description, Preconditions, Run Options, Setup/Cleanup, and Advanced. You use these controls for all operations related to creating or modifying a test, including adding, modifying, copying, and deleting a test.

Basic Operations for Test Editing

This section describes how to add, modify, copy, and delete tests in the Sequence Editor.

Adding a Test

Perform the following steps to add a new test to a sequence:

1. In the Test list box, click on the test that is above or below the position where you want to insert the new test.
2. Click on **New Test** or **New Subseq**. The new, empty test appears in the Test list box.
3. Enter or select the desired values for Test Name, Function Name, File Name, Limit Specification, Input Buffer, Run Options, and Setup/Cleanup. The Test Executive updates the Test list box as you make changes.

Modifying a Test

Perform the following steps to modify a test:

1. Click on the test you want to edit in the Test list box. The name of the test you selected now appears in the Test Name control of the Test Attributes area.
2. Enter or select the desired values for Test Name, Function Name, Input Buffer, Run Options, and Setup/Cleanup.

Copying a Test

To copy a test and insert it into a new position, perform the following steps:

1. Click on the test you want to copy in the Test list box.
2. Click on the **Copy** button or select **Copy** from the **Edit** menu.
3. Slide the Insert Position switch to Above or Below depending on where you want the test to appear in the list.
4. In the Test list box, click on the test that is adjacent to (above or below) the place you want to paste the copied test.
5. Click on the **Paste** button or select **Paste** from the **Edit** menu. The copied test appears above or below the test you selected in the list, depending on how you set the Insert Position switch.

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 or select **Cut** from the **Edit** menu. The selected test disappears.

Sequence Editor Controls and Indicators

This section describes the controls and indicators of the Sequence Editor.

Cut

The **Cut** button copies the test highlighted in the Test list box to the clipboard and deletes the test from the test sequence. You can also select **Cut** from the **Edit** menu.

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. You can also select **Copy** from the **Edit** menu.

Paste

The **Paste** button pastes the test in the clipboard into the Test list box. The value of **Insert Position** determines whether the test appears above or below the currently selected test in the Test list box. You can also select **Paste** from the **Edit** menu.

New Test

The **New Test** button inserts a new, empty test. The test is inserted above or below the currently selected test depending on the value of **Insert Position**. Figure 4-2 shows the test attributes area of the Sequence Editor. This area is a child panel within the larger panel that is the

Sequence Editor. See the *Test Attributes Area* section later in this chapter for more information about editing tests.

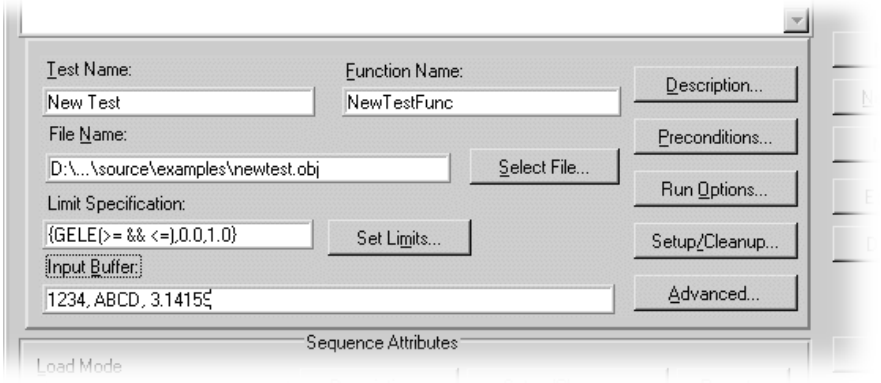


Figure 4-2. Test Attributes Area

New Subseq

The **New Subseq** button inserts a new, empty subsequence. The subsequence appears above or below the currently selected test, depending on the value of **Insert Position**. Figure 4-3 shows the subsequence attributes area of the Sequence Editor dialog box. This area is a child panel within the larger panel that is the Sequence Editor. See the *Test Attributes Area* section in this chapter for more information about editing subsequences.

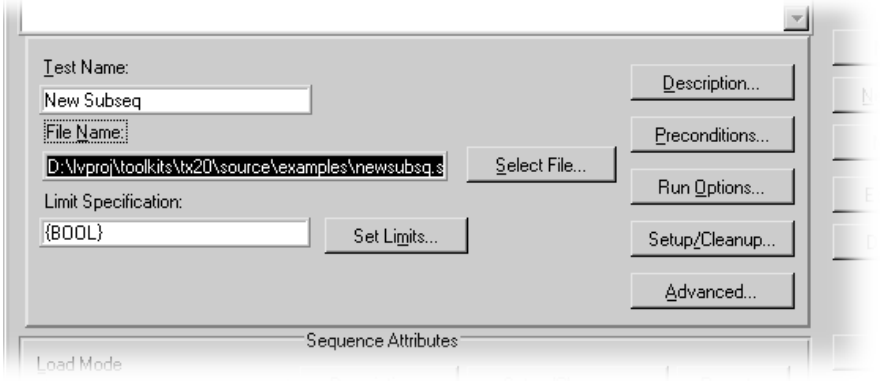


Figure 4-3. Subsequence Attributes Area

New Goto

The **New Goto** button inserts a new, empty goto. The goto appears above or below the currently selected test depending on the value of **Insert Position**. Figure 4-4 shows the goto attributes area of the Sequence Editor. This area is a child panel within the larger panel that is the Sequence Editor.

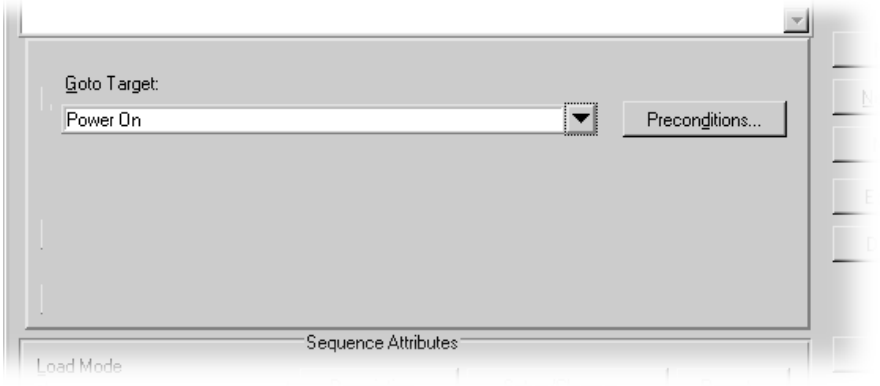


Figure 4-4. Goto Attributes Area

Goto Target

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

Preconditions

This control invokes the precondition editor.

☞ **Note:** ***Like a test, goto statements can have preconditions that determine the conditions under which they execute. You can specify these preconditions in the Precondition Editor that appears when you click on the* Preconditions *button.***

Edit Paths...

The **Edit Paths...** button opens a dialog box where you can change all the paths stored in the sequence. This helps you update the paths when the sequence changes. Figure 4-5 shows the Edit Paths dialog box.

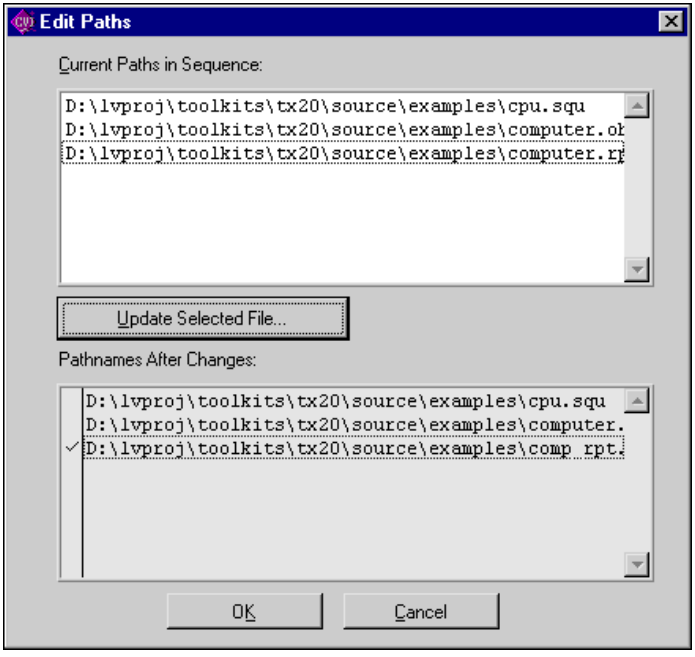


Figure 4-5. Edit Paths Dialog Box

Current Paths in Sequence

This list box shows all the unique pathnames currently in the sequence. To update a path, you can click on a test name in this list box. Then click on **Update Selected File**. A dialog box appears where you can choose an updated path for the file.

Update Selected File...

This button opens a file dialog box where you can specify a new path.

Pathnames after Changes

This list box displays the pathnames the Test Executive uses; it is only an indicator. A checkmark appears to the right of the name(s) when use the **Update Selected File...** button to update a pathname.

Database

The **Database…** button opens a dialog box where you can set options for saving test results to a database. Figure 4-6 shows the Database Options dialog box.

☞ **Note:** ***You must have the LabWindows/CVI SQL Toolkit and use the Test Suite project (*`tstsuite.prj`*) to perform database operations.***

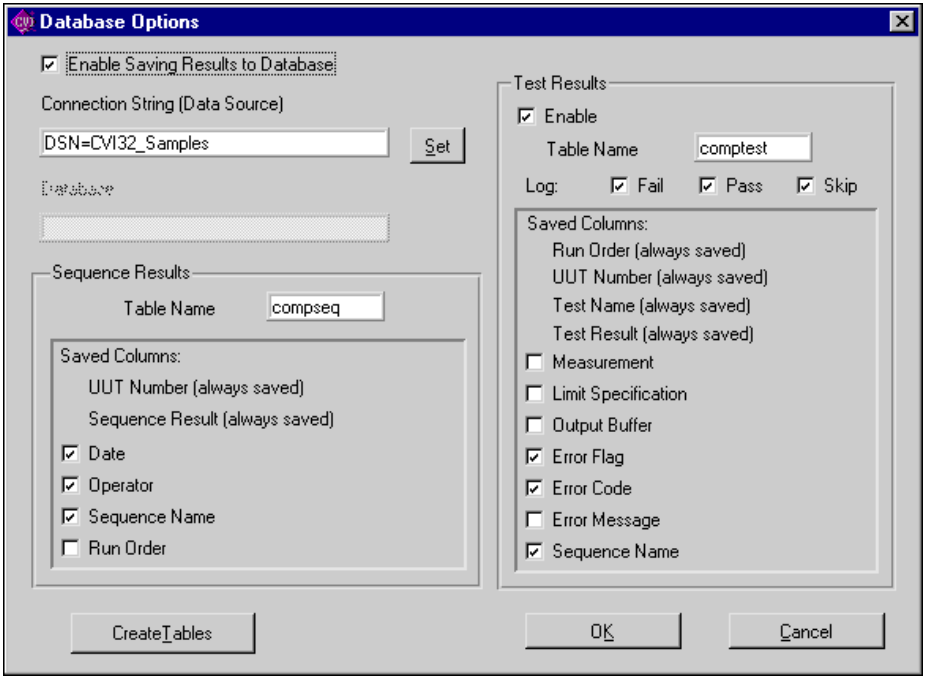


Figure 4-6. Database Options Dialog Box

Enable Saving Results to Database

This check box enables database operations for the sequence.

Connection String (Data Source)

Enter the connection string or use the **Set** button to select from a list of available data source names. See the *LabWindows/CVI SQL Toolkit Reference Manual* for more information on Data Sources. The **Set** button appears only in the Test Suite project, `tstsuite.prj`.

Database

This field identifies the default database for database systems that allow you to store tables in separate databases. In most cases, you do not need to use the **Database** field.

Sequence Results

Controls in this area determine how the Test Executive saves sequence results in the database.

Table Name specifies the database table that stores sequence results.

The checkboxes under **Saved Columns** allow you to choose the information to save for each sequence result. The UUT Number and Sequence Result are always saved. You can also save the Date, Operator, Sequence Name, and Run Order. When you save results from different sequence files in the same table, you may want to save the Sequence Name, too, to make it easier to identify the origin of each result.

Test Results

Controls in this area determine how the Test Executive saves individual test results in the database.

The **Enable** checkbox enables saving of test results to a database table.

Table Name specifies the database table used to store test results.

The **Log** checkboxes allow you to select whether Fail results, Pass results, and Skip results are saved to the test results table.

The checkboxes under **Saved Columns** allow you to choose the information that the Test Executive saves for each test result. The Run Order, UUT Number, Test Name and Test Result are always saved. You can also save the Measurement, Limit Specification, Output Buffer, Error Flag, Error Code, Error Message, and Sequence Name.

Create Tables

The Test Suite automatically creates the database tables as needed. You can use the **Create Tables** button to force the immediate creation of the tables.

OK

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

☞ **Note:** ***The Test Executive saves changes to any database options when you save the sequence file.***

Cancel

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

Test Attributes Area

You use the controls in the Test Attributes area to set the attributes of individual tests in the test sequence. You also use the controls in this area to set attributes of subsequences. This area is a child panel within the larger panel that is the Sequence Editor. Figure 4-7 shows the Test Attributes area.

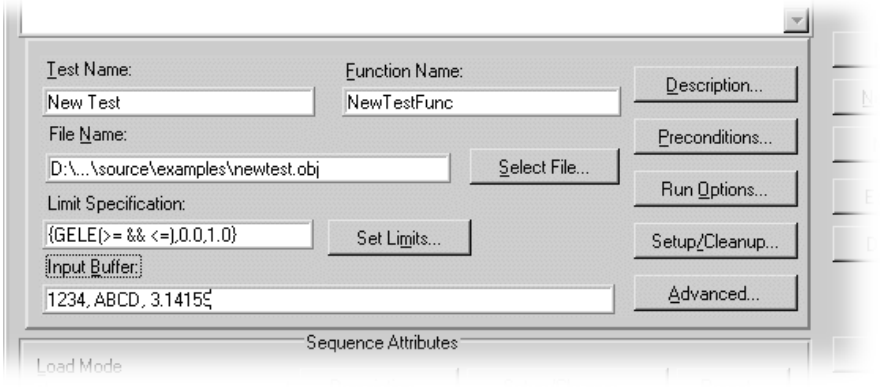


Figure 4-7. Test Attributes Area

Test Name

Type any ASCII string in the Test Name control. The name appears in the Sequence Display of the Test Executive front panel when the sequence is loaded. Each test must have a unique name.

Function Name

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

☞ **Note:** ***Subsequences have no function names.***

File Name

Type the full path name of the file that contains the test into this field. You can also click on **Select File...** to open the File dialog box where you can locate and select the file name.

☞ **Note:** ***National Instruments recommends that you use full pathnames so that the Test Executive can adjust paths when you or a user moves a sequence and its test files. If you use a relative path, the path is defined relative to the Test Executive project or executable, not your sequence file.***

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 field. To specify a limit, click on the **Set Limits...** button. Use the Comparison Type ring control to set the comparison type and the measurement. Figure 4-8 shows the Set Limits for Test dialog box.

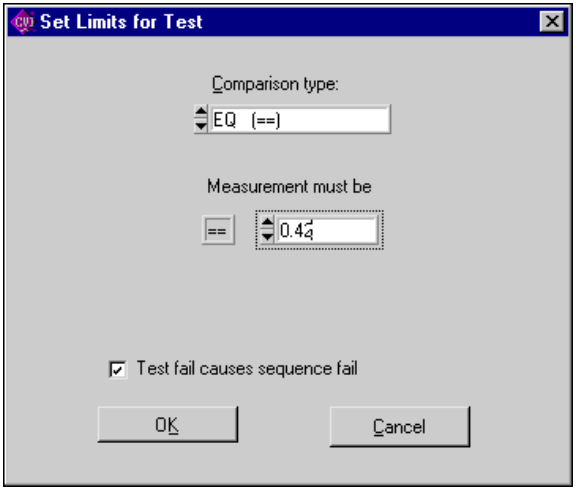


Figure 4-8. Set Limit for Test Dialog Box

Comparison Type ring control specifies the type of comparison to perform, if any, to determine whether a test passed. Table 3-3, *Comparison Type Values*, in Chapter 3, *Operating the Test Executive*, describes the values for each comparison type.

For some comparison type settings, one-limit or two-limit entry controls appear in the Set Limits for Test dialog box. A one-limit value appears for the comparison types `EQ`, `NE`, `GT`, `LT`, `GE`, and `LE`. As shown in Figure 4-9, two limit values, a lower and upper limit, appear for the comparison types `GTLT`, `GELE`, `GELT`, and `GTLE`. No limits appear for the comparison types `BOOL`, `LOG`, and `NONE`.

☞ **Note:** ***Subsequences can use only the comparison types*** `BOOL`**,** `LOG`***, and*** `NONE`**.**

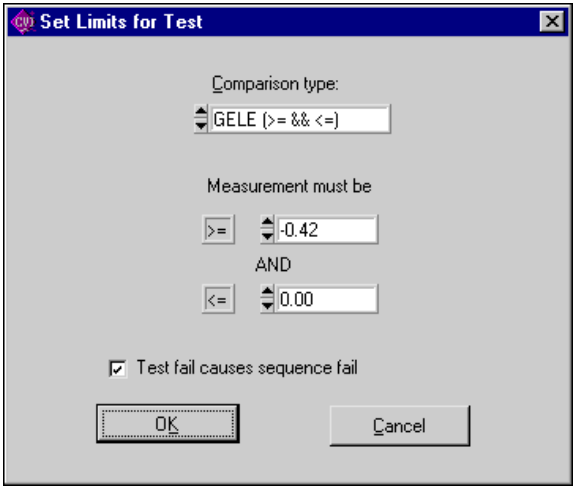


Figure 4-9. Comparison Type Settings

Input Buffer

As shown in Figure 4-7, the Input Buffer control displays 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.

☞ **Note:** ***Subsequences have no input buffers.***

Description

The **Description** button, shown in Figure 4-7, invokes the Test Description dialog box where you can enter a description of an individual test.

Preconditions

The **Preconditions** button, shown in Figure 4-7, invokes the Precondition Editor. See the *Editing Preconditions* section in this chapter for more information on the Precondition Editor.

Run Options

The **Run Options** button, shown in Figure 4-7, opens the Test Run Options dialog box where you can specify the Run Mode, Fail Action, Pass Action, and the maximum number of loops for a test. Figure 4-10 shows the Test Run Options dialog box.

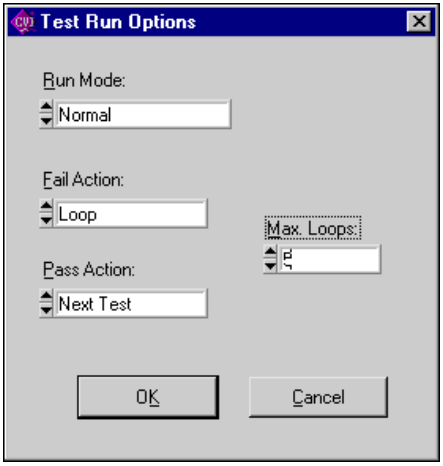


Figure 4-10. Test Run Options Dialog Box

Run Mode

Run mode specifies how the test executes. Table 4-3 describes the options for Run Mode.

Table 4-3. 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. Table 4-4 describes the options for Fail Action.

Table 4-4. 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. Table 4-5 describes the options for Pass Action.

Table 4-5. 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. This control 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, shown in Figure 4-7, invokes the Test Setup/Cleanup Routines dialog box for an individual test. Figure 4-11 shows the Test Setup/Cleanup Routines dialog box.

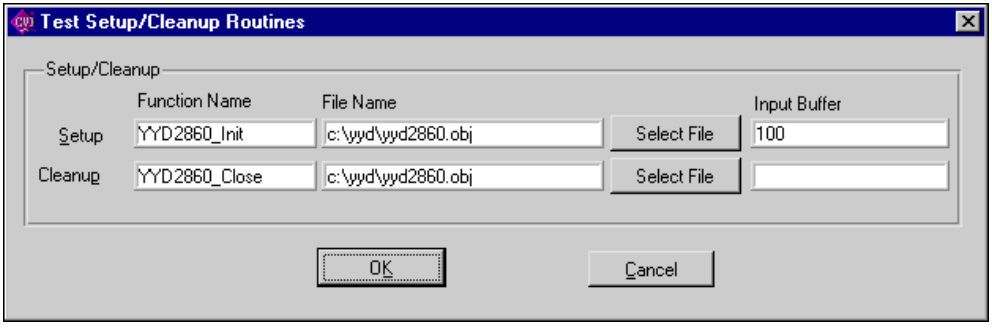


Figure 4-11. Test Setup/Cleanup Routines Dialog Box

Setup Function

A setup function executes before a test. You enter the name of the setup function in the Function Name control. You enter the name of the file that contains 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 you leave the Setup-Function Name control blank, no setup function runs. See the *Writing Test Functions* section of this chapter for information about writing your setup function.

Cleanup Function

A cleanup function executes after a test. Enter the name of the cleanup function in the Function Name control. Enter the name of the file that contains 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 you leave the Cleanup-Function Name control blank, no cleanup function runs. See the *Writing Test Functions* section of this chapter information about writing your cleanup function.

Advanced

The **Advanced** button, shown in Figure 4-7, invokes the Advanced Test/Subsequence Attributes dialog box for an individual test or subsequence. Figure 4-12 shows this dialog box.

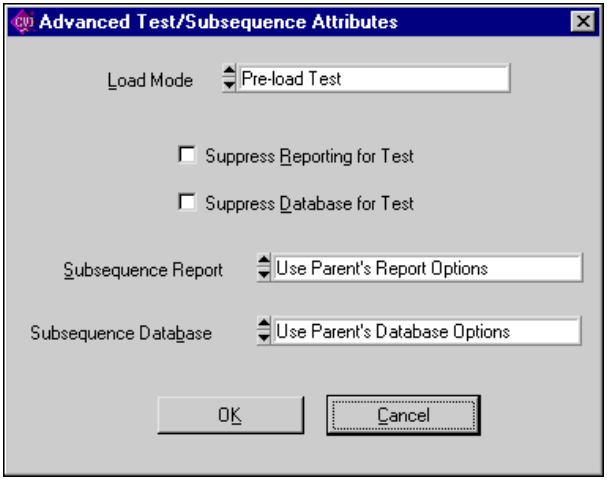


Figure 4-12. Advanced Test/Subsequence Attributes Dialog Box

Load Mode

Load Mode specifies the load mode for an individual test. The load mode can be either Pre-Load Test or Dynamic Load Test. To make these values take effect, you must set the Load Mode ring control for sequences (which is located in the Sequence Attributes area of the Sequence Editor dialog box) to `Use Test Load Specs`.

Suppress Reporting for Test

When selected, Suppress Reporting for Test prevents the Test Executive from saving any report information in the report file regarding the test you are modifying.

Suppress Database for Test

When selected, Suppress Database for Test prevents the Test Executive from writing test result information to the database regarding the test you are modifying.

Subsequence Report

Subsequence Report sets the report behavior within a subsequence. While a subsequence executes, it can use one of the following command options from this ring control:

- Use Parent's Report Options
- Use Subsequence's Report Options
- Suppress Reporting within Subsequence

Subsequence Database

Subsequence Database allows you to select the database behavior of a subsequence.While a subsequence executes, it can use one of the following command options from this ring control:

- Use Parent's Database Options
- Use Subsequence's Database Options
- Suppress Database within Subsequence

Sequence Attributes

The controls in the Sequence Attributes area are used to specify the load mode, description, setup/cleanup functions, and report file for the test sequence. You can see all these controls in the Sequence Editor dialog box shown in Figure 4-1.

Load Mode

Load Mode controls how the sequence is loaded. You can select one of the following items in the Load Mode ring control:

- **Pre-Load Tests**—All tests and subsequences are loaded when the sequence is loaded.
- **Dynamic Load Tests**—Tests and subsequences are loaded and unloaded as needed.
- **Use Test Load Specs**—The load specification of each test determines whether it is loaded when the sequence is loaded or loaded only as needed.

Description

The **Description** button displays a dialog box where you can enter and modify the description of the test sequence. Figure 4-13 shows this dialog box. The test sequence description appears in the Test Report that the Test Executive generates when it executes a test sequence. The first line of the description also appears in the Description field on the Test Executive front panel.

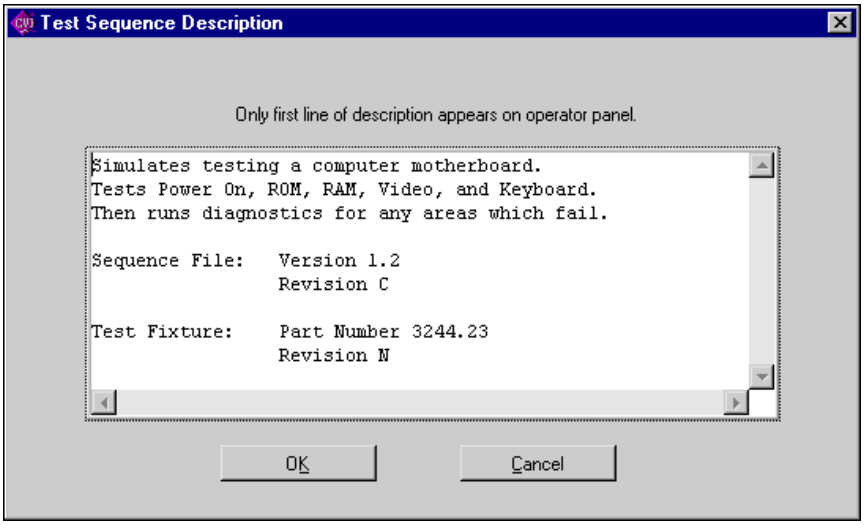


Figure 4-13. Test Sequence Description Dialog Box

Setup/Cleanup

The **Setup/Cleanup** button, shown in Figure 4-1, opens the Sequence Setup/Cleanup Routines dialog box. There are two types of setup/cleanup functions available. Sequence Execution functions run at the beginning and at the end of each sequence execution. Sequence

load/unload functions run when the sequence is loaded or unloaded. Figure 4-14 shows the Sequence Setup/Cleanup Routines dialog box.

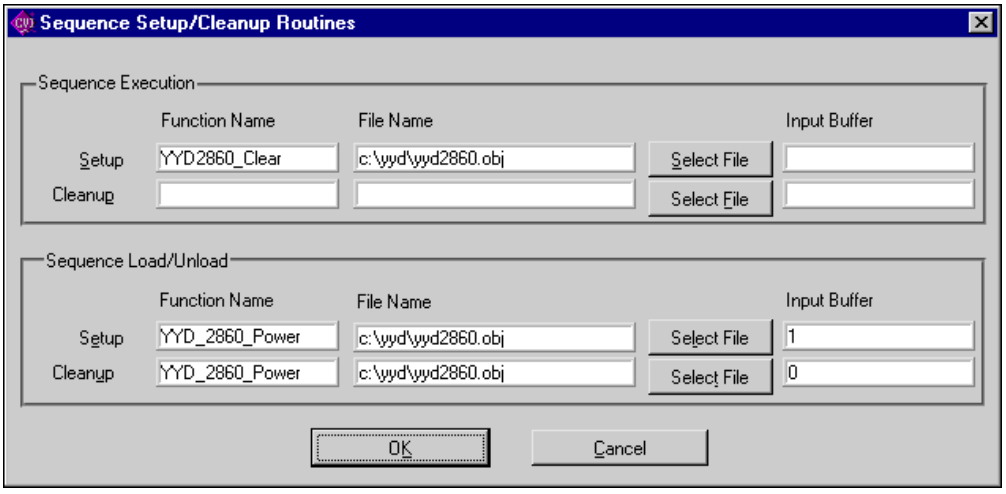


Figure 4-14. Sequence Setup/Cleanup Routines Dialog Box

Report...

The **Report...** button, shown in Figure 4-1, displays a dialog box where you set the attributes of your Test Report. Figure 4-15 shows the Set Default Report File dialog box. You type the name of the report file you want to create in the Test Report File control or 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 adds 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.

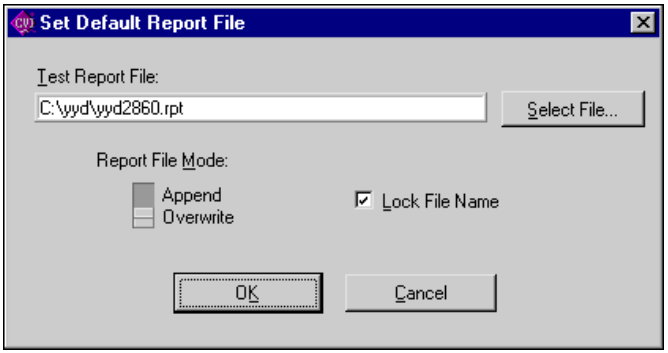


Figure 4-15. Set Default Report File Dialog Box

Editing Preconditions

The preconditions of a test specify what other tests must pass or fail before that particular test executes. To define the preconditions of a test, click on the **Preconditions** button, shown in Figure 4-1, in the Test Attributes dialog box of the Sequence Editor. Figure 4-16 shows the Precondition Editor.

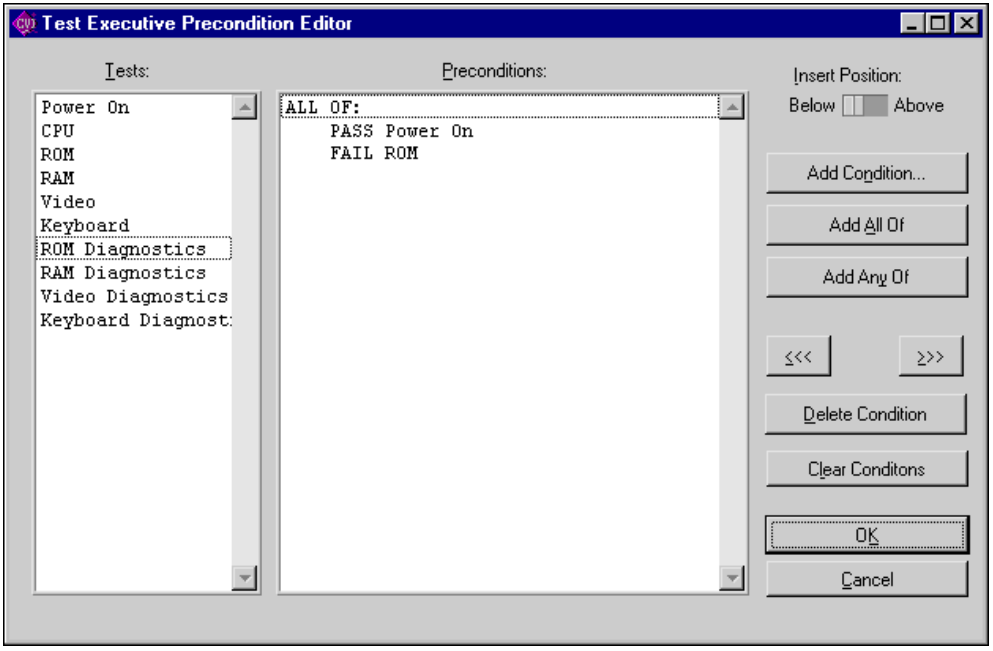


Figure 4-16. Precondition Editor

The name of each test in the test sequence appears in the Tests list box. 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 whichever test name you select in the Tests list box. In Figure 4-16, Power On test and ROM test are the preconditions for ROM Diagnostics. ROM Diagnostics runs only when Power On test passes and ROM test fails.

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

1. In the Tests list box click on the test for which you want to set preconditions. All the preconditions you set will apply to the test you have selected.

2. If you want to apply multiple preconditions, establish grouping for the preconditions by clicking on the appropriate button:
   a. Click on **Add All Of** to insert `ALL OF:` in the Preconditions list box. This type of heading introduces a series of preconditions that must all be true in order for the test to run.
   b. Click on **Add Any Of** to insert `ANY OF:` in the Preconditions list box. This type of heading introduces a series of preconditions of which at least one must be true.
   c. You can nest `ALL OF:` and `ANY OF:` headings, to create more complex preconditions.
3. Click on **Add Condition...**. The Add Condition dialog box shown in Figure 4-17 appears.

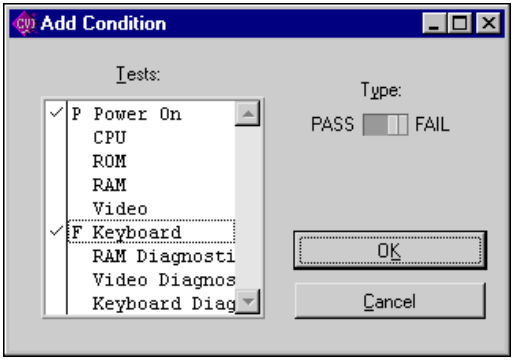


Figure 4-17. Add Condition Dialog Box

4. In the Add Condition dialog box, set the Type switch to PASS or FAIL, depending on whether you want the precondition test(s) to pass or fail.
5. Select the precondition test(s) from the list. A checkmark appears beside the test and a `P` (pass) or an `F` (fail) also appears, depending on whether you have set the Type switch to PASS or FAIL.
6. Click on **OK** to confirm your settings and return to the Sequence Editor dialog box.

The following list describes all the controls in the Precondition Editor dialog box:

- The Insert Position switch determines whether new preconditions are 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 invokes the Add Condition dialog box.
- The Tests list box shows the available precondition tests. The setting of the Type switch (pass or fail) determines whether the precondition test(s) you select must pass or fail.
- The **Move to the Left** and **Move to the Right** buttons, shown in Figure 4-18, 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 4-18. 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 the preconditions for the test selected in the Tests list box.
- The **OK** button saves any changes you make 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. The Test Executive saves all preconditions with the test sequence when you select **Save** from the **File** menu on the Test Executive front panel to save the test sequence to disk.
- The **Cancel Edits** button discards any changes you make to the test sequence preconditions and returns you to the Sequence Editor.

Effect of Preconditions and Run Mode on 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:

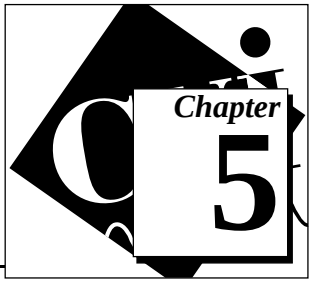
1. The Text Executive evaluates the preconditions for the test. For a test to execute, the result of each precondition test must match the result specified in the preconditions. If evaluation of preconditions indicates that the current test should be skipped, the test result is set to `SKIP`.

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 4-6 shows the value that each run mode generates for a skipped test.

Table 4-6. Run Mode Test Result Values

Run Mode	Value Generated
Normal	Test success determines value
Skip (Test not run)	`SKIP`
Force PASS (Test not run)	`PASS`
Force FAIL (Test not run)	`FAIL`

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



Modifying the Test Executive

This chapter describes the organization and internal structure of the Test Executive and suggests where you can make modifications to the behavior of the Test Executive. The chapter covers the following topics:

- Test Executive Overview
- Organization of source code files
- Common modifications

Test Executive Overview

To customize the Test Executive, it is important that you understand how the Test Executive Engine works and how the Test Executive projects are organized.

Test Executive Projects

Consider how a typical Test Executive project interacts with the Test Executive Engine. The Test Executive Engine handles most of the sequence and test operations, but the engine does not have any user interfaces, except the sequence editor and the database viewer. The Test Executive project handles all other user interface interaction, processing user events on the Test Executive front panel and dialog boxes. The Test Executive project user interface can include the display of loaded sequences and their tests, execution status, pass/fail banners,

and login dialog boxes. Figure 5-1 outlines the functional areas of the Test Executive project and the Test Executive Engine.

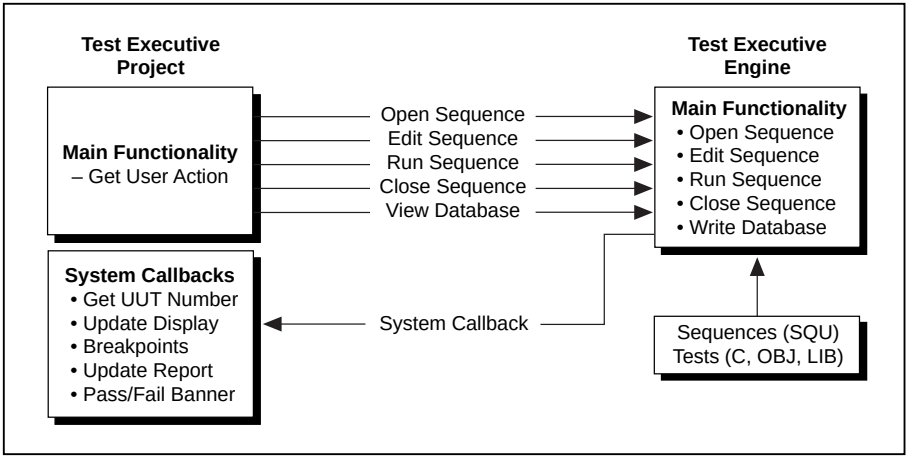


Figure 5-1. Functional Description of Test Executive Projects

Test Executive Engine

The Test Executive Engine performs the following tasks:

- Opening and closing sequences
- Editing and saving sequences
- Executing sequences and tests
- Error Handling
- Writing results to Test Executive ODBC (Open Database Connectivity) databases

System Callbacks

During the execution of an engine operation the Test Executive Engine uses *system callbacks* to communicate interim status to the calling application. For example, when the engine is executing a request to run a sequence, the engine can notify the calling application before and after the execution of each test in the sequence. These callbacks allow the application to update its user interface to indicate the status of each test as they execute, such as *running*, *pass*, or *fail*. System callback functions are registered with the Test Executive Engine by means of the

`SetEngineAttribute` function. Table 5-1 lists system callbacks and their corresponding engine attributes.

Table 5-1. System Callbacks

Type	Engine Attribute
Running a test	`TXATTR_BEFORE_RUNTEST_FUNC_PTR` `TXATTR_AFTER_RUNTEST_FUNC_PTR`
Running a sequence	`TXATTR_BEFORE_RUNSEQ_FUNC_PTR` `TXATTR_AFTER_RUNSEQ_FUNC_PTR`
Testing a set of UUTs	`TXATTR_BEFORE_UUTLOOP_FUNC_PTR` `TXATTR_AFTER_UUTLOOP_FUNC_PTR`
Opening a sequence	`TXATTR_OPENSEQ_FUNC_PTR`
Closing a sequence	`TXATTR_CLOSESEQ_FUNC_PTR`
Saving a sequence	`TXATTR_SAVESEQ_FUNC_PTR`
Entering a breakpoint	`TXATTR_BREAKPOINT_FUNC_PTR`

The system callback function adheres to the following prototype:

`int CVICALLBACK` **SystemCallback** (`int` **callbackEvent**, `int` **seqId**, `int` **runTestType**, `int` **resultId**);

The **callbackEvent** parameter specifies the type of callback. This parameter is useful because it permits you to use a single function for more than one type of *system callback*. Valid events:

- `TXRE_BEFORE_RUNTEST`
- `TXRE_AFTER_RUNTEST`
- `TXRE_BEFORE_RUNSEQ`
- `TXRE_AFTER_RUNSEQ`
- `TXRE_BEFORE_UUTLOOP`
- `TXRE_AFTER_UUTLOOP`
- `TXRE_OPENSEQ`
- `TXRE_SAVESEQ`
- `TXRE_CLOSESEQ`
- `TXRE_BREAKPOINT`

The **seqId** parameter specifies the sequence identifier or handle associated with the system callback.

The **runTestType** parameter specifies the type of execution, such as one of the following types:

- `TXRT_LOOP_SEQ`
- `TXRT_SINGLE_PASS`
- `TXRT_LOOP_TEST`
- `TXRT_SINGLE_TEST`

The **resultId** parameter specifies the handle to the result record that the program can access using the `TX_GetResultAttribute` function.

Process Model

The following diagrams illustrate how the Test Executive Engine executes tests in a sequence and when each type of system callbacks is called.

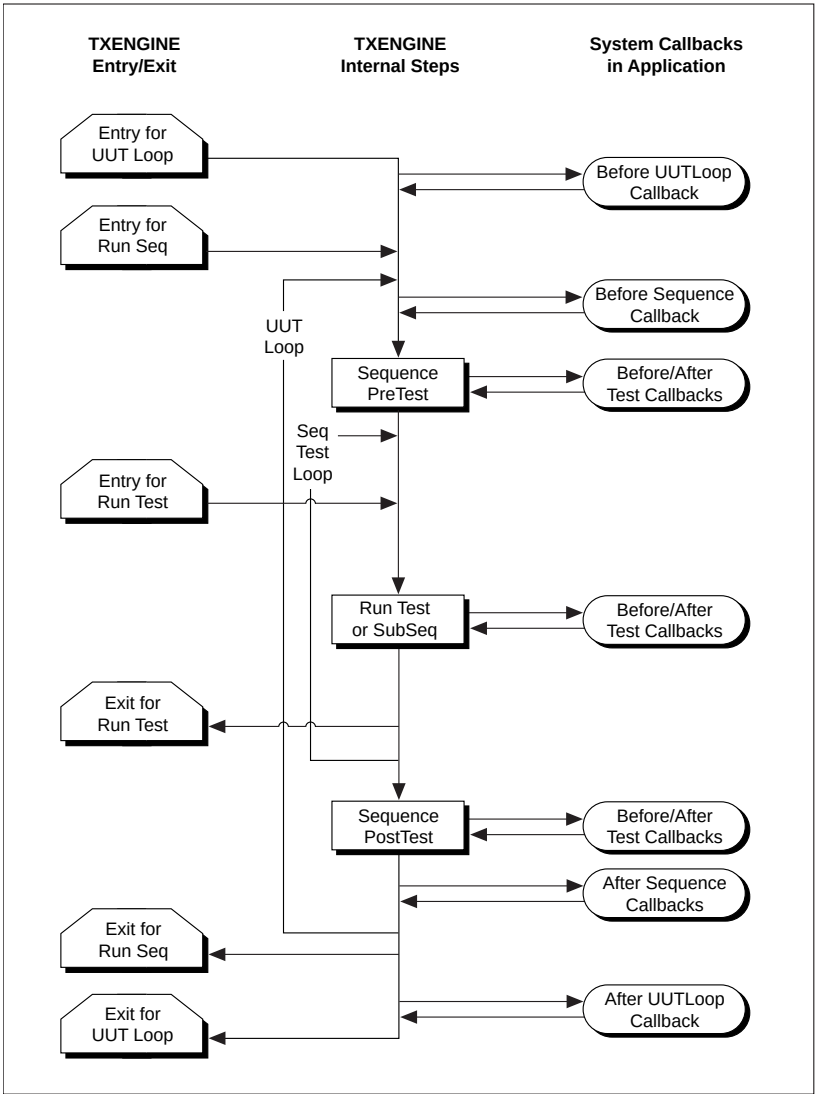


Figure 5-2. System Callbacks for Running a Sequence

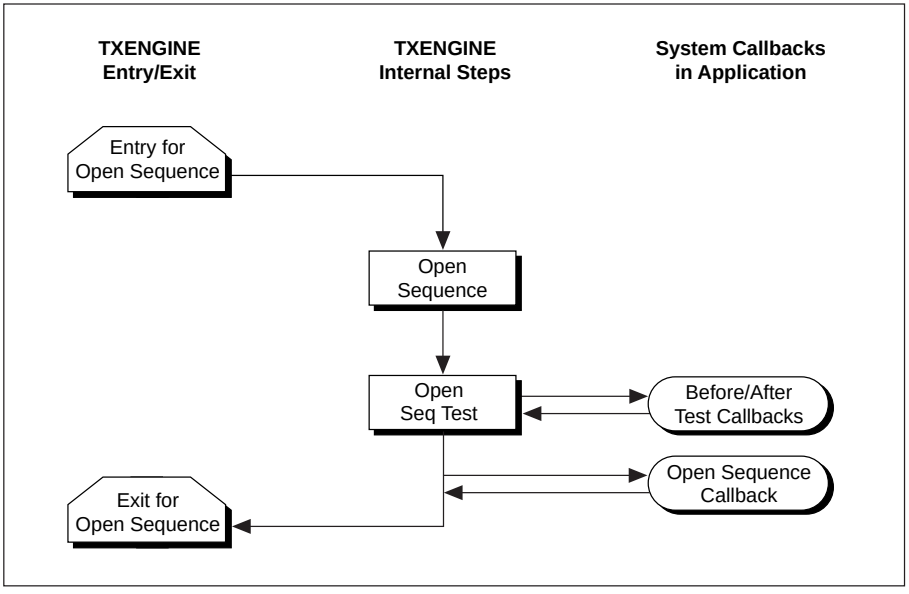


Figure 5-3. System Callbacks for Opening a Sequence

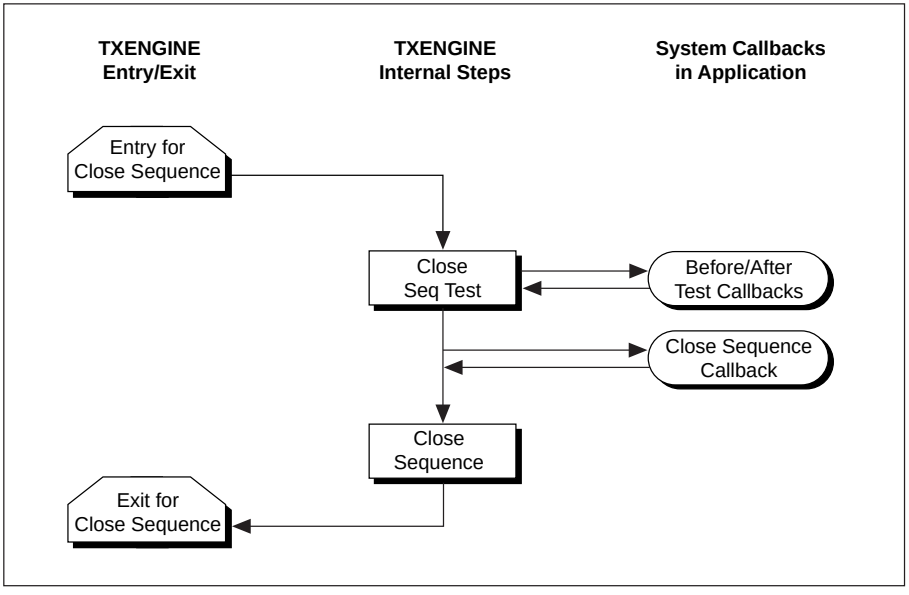


Figure 5-4. System Callbacks for Closing a Sequence

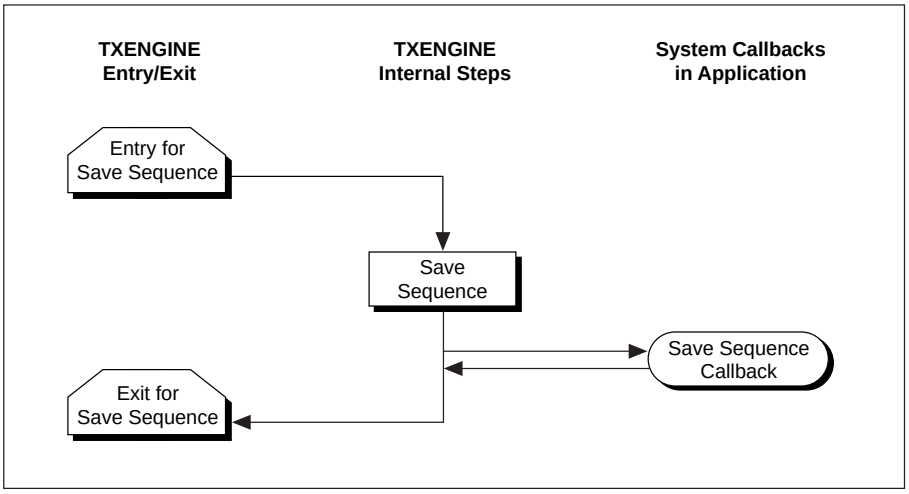


Figure 5-5. System Callbacks for Saving a Sequence

Organization of Source Files

The Test Executive contains several source code files. Source files are divided into *GUI* (graphical user interface) files and *engine* files. The GUI files control the front panel user interface and contain the higher-level functions which you can modify to customize the Test Executive behavior. The engine files contain the core of the Test Executive process model. In most cases you do not need to modify the engine files to customize the Test Executive.

Figure 5-6 shows the directory structure for the LabWindows/CVI Test Executive Toolkit.

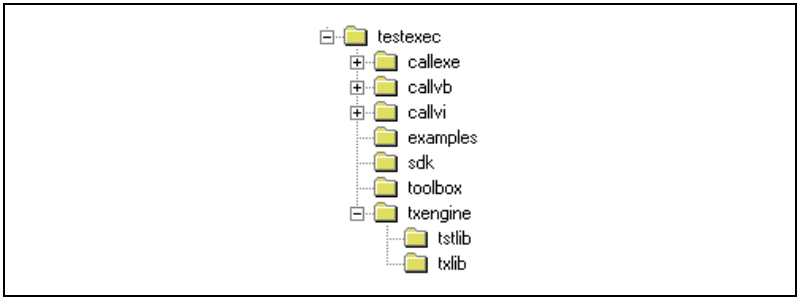


Figure 5-6. Directory Structure of the LabWindows/CVI Test Executive

The base directory, `testexec`, contains several subdirectories (described in Table 5-2) and the Test Executive project files, source code files, and user interface files.

Table 5-2. Directories Contained in the Base Directory of the Test Executive

Directory Name	Contents
`callexe`	CallExe add-on (Windows only). See the `callexe.hlp` file for more information.
`callvb`	CallVB add-on (Windows 95/NT only). See the `callvb.hlp` file for more information.
`callvi`	CallVI add-on (Windows 95/NT only). See the `callvi.hlp` file for more information.
`examples`	Example sequence files and test source code.
`sdk`	SDK files for LabWindows/CVI Base Package (Windows 95/NT only).
`toolbox`	Interim toolbox instrument drivers for LabWindows/CVI version 3.1. (See `readme.txt` for more information.)
`txengine`	Test Executive Engine library and source files.

Source Files for the Test Executive Projects

The Test Executive project (`.prj`) files and related source (`.c`) files are located in the base installation directory, `testexec`. Table 5-3 describes each file in this directory.

Table 5-3. List of Files in the Test Executive Directory

File Name	Description
`testexec.prj`	Test Executive project
`tstsuite.prj`	Test Executive project with database connectivity
`txeditor.prj`	Standalone Test Executive editor

Table 5-3. List of Files in the Test Executive Directory (Continued)

File Name	Description
tseditor.prj	Standalone Test Executive editor with database connectivity
txext.prj	Object modules for external compiler support
txengine.lib txengine.fp txengine.h	Test Executive Engine Library, located in the \txengine\ subdirectories
txgui.uir	User interface for the front panel
txmain.c	Initialization source code for the engine and user interface
txedsqmn.c	Initialization source code for the engine (standalone editor only)
txguiutl.c	Support source code for user interface
txguicb.c	User interface callbacks for main panels
txusrutl.c	Source code for front-end utility module
txsyscb.c	System callbacks for running, open/close/save, and breakpoints
txloginu.uir	Login user interface
txlogin.c	Login code
txreport.c	Source code for report filing
txrtview.c	Source code for internal report viewer
txrtmgr.c	Source code for report file manager

testexec.prj and tstsuite.prj are the project files for the Test Executive. The Test Suite Test Executive, tstsuite.prj, has the ODBC database features enabled.

`txeditor.prj` and `tseditor.prj` are the project files for the standalone Test Executive Editor. The Test Suite Editor, `tseditor.prj`, has the ODBC database features enabled.

`txext.prj` is a project file used to create files to help you compile the Test Executive projects in external compilers under Windows 95/NT. See the section, *Using External Compilers*, in this chapter for more information.

`txengine.fp`, `txengine.lib`, and `txengine.h` are the Test Executive Engine Library files. There are two versions of these files for Windows. The `\txengine\txlib\` version does not connect with databases. The `\txengine\tstlib\` version does connect with databases. See the section *Requirements for Database Connectivity* in this chapter for more information on using the database options.

`txgui.uir` contains the user interface for the front panel of the Test Executive.

`txmain.c` contains the function `main()`, which handles initialization of the Test Executive Engine and the Test Executive user interface.

`txedsqmn.c` contains the function `main()` for the standalone editor, which handles initialization of the Test Executive Engine and the launching of the user interface for editing sequences.

`txguiutl.c` and `txguicb.c` control the user interface for the main panel of the Test Executive. The principal user interface callbacks in `txguicb.c` are `MainPanelCallback` and `MainMenuCallback` which handle most of the menus and controls on the Test Executive front panel.

`txusrutl.c` contains miscellaneous functions to support `txguicb.c` and `txguiutl.c`.

`txsyscb.c` contains the system callbacks. These callbacks allow you to modify the behavior of the Test Executive at specific times such as when a sequence loads or before a setup function for a test runs. You can modify the existing system callbacks or edit `SetupRunOptions` to replace the existing callbacks with new callback functions. The default system callback functions are `UserRunCallback` for running sequences or tests, `UserOpenCloseSaveCallback` for opening, closing or saving sequences, and `UserBreakpointCallback` for breakpoints.

txloginu.uir contains the user interface panels for user login.

txlogin.c handles login by users.

txreport.c handles test report generation. The major functions are WriteRptHeader, WriteSeqHeader, and WriteTestResult. The Test Executive keeps a report buffer for each report file. (Remember that a single report buffer may be used by multiple sequences).

txrtmgr.c handles management of the report buffer, including the opening, writing and closing of report files.

txrtview.c implements the internal report viewer.

Source Files for the Test Executive Engine

The Test Executive Engine project (.prj) files and their source code (.c) reside in the \txengine subdirectory. Other file types in this subdirectory include batch (.bat), user interface (.uir), and makefile (.mak). Table 5-4 describes each file of the Test Executive Engine.

Table 5-4. List of Files in the Test Executive Engine

File Name	File Description
txlib.prj	Project file to build the engine without database connectivity in \txlib\txengine.lib (Windows 95/NT only)
tstlib.prj	Project file to build the engine with database connectivity in \tstlib\txengine.lib (Windows 95/NT only)
makew16.bat	Batch file to build txengine.lib files using an external WATCOM compiler/linker (Windows 3.1 only)
txengine.mak	Makefile to build txengine.a library file using an external UNIX compiler/linker (Sun and HP-UX only)
txcore.c	Source code for the lowest level core of the engine
txloadsq.c	Source code for loading sequences and tests
txruntst.c	Source code for running sequences and tests

Table 5-4. List of Files in the Test Executive Engine (Continued)

File Name	File Description
txprecnd.c	Source code for evaluating preconditions
txedseq.c	Source code for sequence editor
txdbed.c	Source code for database options editor
txedpc.c	Source code for precondition editor
txdbsupp.c	Source code for database access manager
txsavres.c	Source code for test result manager
txerrors.c	Source code for error handling manager
txutil.c	Source code for utility functions
txedsequ.uir	Sequence editor user interface
txdbuir.uir	Database viewer user interface

`txlib.prj` creates a version of the Test Executive Engine without database connectivity in the `\txengine\txlib\`subdirectory. Both the `testexec.prj` and `txeditor.prj` projects use this version of the engine. (Windows 95/NT only)

`tstlib.prj` creates a version of the Test Executive Engine with database connectivity in the `\txengine\tstlib\`subdirectory. Both the `tstsuite.prj` and `tseditor.prj` projects use this version of the engine. See the section, *Requirements for Database Connectivity*, in this chapter for more information on using this version of the engine. (Windows 95/NT only)

`makew16.bat` creates the Test Executive Engine static library using the WATCOM external compiler/linker (Windows 3.1 only)

`txengine.mak` creates the Test Executive Engine static library using an external UNIX compiler/linker (Sun and HP-UX only)

`txcore.c` contains the low-level functions that maintain, load, and save sequences and that run tests.

`txloadsq.c` contains the high-level functions for loading and saving sequences.

`txruntst.c` contains high-level functions that run tests and handle loading of subsequences and tests during the running of a test sequence. This source code file also contains functions that process breakpoints.

`txprecnd.c` handles evaluation of preconditions.

`txedseq.c` loads and operates the Sequence Editor panels and its popup dialog boxes.

`txdbed.c` handles editing of the database options for the Sequence Editor. The user interface callback for the Database Options panel is `DBOptsCallback`.

`txedpc.c` handles the Precondition Editor panel and its pop-up dialog boxes.

`txdbsupp.c` contains the routines that support database access.

`txsavres.c` maintains a record of current and previous test results. The Test Executive Engine uses the results for the current sequence when evaluating preconditions and when saving results to a database. The Test Executive project retrieves test results from the engine to update its display and write data to the sequence report files.

`txerrors.c` handles the error reporting routines.

`txutil.c` contains utility functions that maintain a list of files in a menu and that save options to an INI file or the Windows Registry.

`txedsequ.uir` contains the Sequence Editor user interface panels.

`txdbuir.uir` contains the Database View user interface panels.

Requirements for Database Connectivity

Under Windows, there are two versions of the Test Executive Engine; one with, and one without database connectivity. The Test Suite version, `tstsuite.prj`, includes database connectivity and requires that you link the LabWindows/CVI SQL Toolkit into the project and set the define `ACTIVATE_SQL_TOOLKIT_FUNCTIONS` equal to `1`. (You make this setting in the dialog box that appears when you select **Compiler Defines...** from the **Options** menu of the Project window in LabWindows/CVI.)

Common Modifications

In most cases you do not need to modify the engine files to customize the Test Executive. The most common types of modification affect only the user interface (`.uir`) and source (`.c`) files located in the base directory of the Test Executive.

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

- Login
- Default directories
- Pass, fail, and abort banners
- UUT serial number prompt
- Test reports
- Locking out other applications

Login

For more information on the privileges for each of these login levels see the *Operating Levels* section in Chapter 1, *Introduction*.

Login Dialog on Startup

You can specify whether the login dialog appears on startup of the Test Executive by changing the value of the define `LOGIN_AT_START` in `txguiutl.h`. The default value is `1` which causes the Login dialog box to display.

Default Login Level and Name

You can specify the default login level and operator upon startup of the Test Executive by changing the value of the defines `LOGIN_DEFAULT_LEVEL` and `LOGIN_DEFAULT_NAME` in `txguiutl.h`. The default values are `NONE` and `""`, respectively.

Changing Passwords

You can set the passwords that determine the operating level (Developer, Technician, and Operator) by modifying the function `CheckLogin` in `txlogin.c`. Passwords in `CheckLogin` are case sensitive.

Default Directory

You can specify the default directory that the Test Executive uses for locating sequences and report files by changing the value of the define `DEFAULT_DIR` in `txguiutl.h`. The default value, `""`, specifies the current working directory as the location for sequences and report files.

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, red, and yellow respectively), an **OK** button, and a large label containing a pass, fail, or abort message. These banners are defined in `txgui.uir`.

Changing the UUT Serial Number Dialog

You can modify the dialog box that prompts for the UUT serial number by modifying the UIR panel in `txgui.uir`. You can also make other modifications to the dialog box, such as adding a routine that reads the serial number from a bar code reader through an RS-232 port. To make this type of change you need to make appropriate changes to the function `GetUUTInformationDialog` in `txsyscb.c`.

Changing the Test Report

You can modify the test report format by modifying the module `txreport.c`. The report module contains the following formatting functions:

- `WriteRptHeader`
- `WriteSeqHeader`
- `WriteTestResult`
- `WriteStopReason`
- `WriteSeqExecutionMsg`
- `WritePrePostOutBuffer`

The following example shows a typical 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
```

Use of the “hook” Parameter

The Test Executive Engine uses the following two data structures—tTestData and tTestError—to pass data to and from a test:

```
typedef struct ClassData_Rec {
   Status     result;       /* Whether test passed  */
   double    measurement;    /* Measurement taken    */
                             /* by test function     */
   char *    inBuffer;       /* For passing          */
                             /* parameters into test */
   char *    outBuffer;      /* For output messages  */
                             /* from the test        */
   char *    modPath;        /* Path of module       */
                             /* containing test      */
   char *    modFile;        /* Base file of module  */
                             /* containing test      */
   void *    hook;          /* User defined          */
                             /* expansion hook       */
   int       hookSize;       /* Number of bytes hook */
                             /* points to            */
   tMallocPtr mallocFuncPtr;/* malloc function to    */
                             /* use when mallocing   */
                             /* inBuffer, outBuffer, */
                             /* errorMessage         */
   tFreePtr   reeFuncPtr;   /* free function to use  */
                             /* when freeing for     */
                             /* inBuffer, outBuffer, */
                             /* errorMessage         */
} tTestData;

typedef struct ClassError_Rec {
Boolean   errorFlag;      /* Whether error occurred  */
                          /* in test function        */
tErrLoc   errorLocation;  /* Where error occurred    */
                          /* (pretest, test etc.     */
int       errorCode;      /* User defined error code */
char      *errorMessage; /* User defined error       */
                          /* message                 */
} tTestError;
```

The default test executive projects do not define the **hook** and **hookSize** parameters; their default values are zero. You can define these parameters by setting the attributes TXATTR_HOOK_PTR and TXATTR_HOOK_SIZE through a call to the SetEngineAttribute

function. You can use the hook parameters to pass a special integral value or a pointer to a block of memory to all tests. Each test could reference or even update the value or memory space, and then the application could reference this information in a system callback after the test completes.

☞ **Note:** ***Use of the* hook *and* hookSize *parameters is specific to the this toolkit and may be incompatible with future versions of Test Executives.***

Lock Out Other Applications

You can specify whether the Test Executive tries to lock out other applications by changing the value of the define `LOCKOUT_OTHER_APPS` in `txguiutl.h`. The default value is to not lock out other applications. See the Utility Library function `DisableTaskSwitching` for further discussion on locking out other applications under the Windows operating system.

Using External Compilers

In order to use an external compiler under Windows 95 and NT, you must build both the Test Executive Engine projects and the Test Executive projects with your external compiler.

For more information on using user interface (`.uir`) files and `LoadExternalModule` in external compilers, see the section *LabWindows/CVI Libraries in External Compilers* in Chapter 3, *Windows 95 and NT Compiler/Linker Issues*, in the *LabWindows/CVI Programmer Reference Manual*.

Rebuilding the Test Executive Engine Library

To use the Test Executive Engine in an external compiler, you must recompile the static library `txengine.lib` to use the ANSI-C libraries of the external compiler.

User Interface Callback and LoadExternalModule Objects

You use the `txext.prj` in the base directory of the Test Executive to create support module files for compiling the Test Executive projects in external compilers. The Test Executive projects need to load the UIR files, and load and link the test object modules when executed. You

should use the LabWindows/CVI External Compiler Support dialog box in this project to create the following object modules.

- User Interface Callback Object File (`txextuir.obj`)
- LoadExternalModule Symbol Object File (`txextref.obj`)

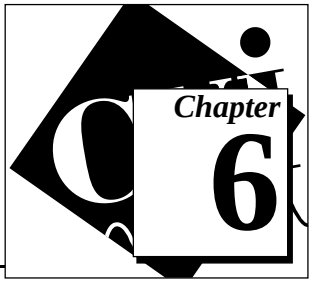
You must add these object files to the external compiler project or workspace to allow the LabWindows/CVI Run-time Engine to load the UIR files and link the test objects properly.

☞ **Note:** ***When you add user interface resource (*** `.uir` ***) files to your Test Executive projects and you wish to compile those projects using an external compiler, you must include those additional*** `.uir` ***files in the*** `txext.prj` ***project file.***

Toolbox and Inifile Instrument Drivers

The Test Executive Engine project uses both the `toolbox` and `inifile` instrument drivers. You need to add the source files for these instrument drivers to the project file for the external compiler.

Distributing the Test Executive



This chapter describes the files required when you distribute a Test Executive executable.

Distribution Overview

Project Files

The distribution kits for the following project files already contain the appropriate files for distribution. You must add to the distribution kit any files that you add to these applications.

- `testexec.prj` (Test Executive without database connectivity)
- `tstsuite.prj` (Test Executive with database connectivity)
- `txeditor.prj` (Test Executive Editor without database connectivity)
- `tseditor.prj` (Test Executive Editor with database connectivity)

Files to Include with Distribution

This section presents the types of files that you distribute with a Test Executive executable: executable, user interface, DLL, and database connectivity files.

Test Executive Executable Files

Each Test Executive project creates a different executable. You can include more than one executable with a distribution.

- `testexec.prj` creates `testexec.exe`
- `tstsuite.prj` creates `tstsuite.exe`
- `txeditor.prj` creates `txeditor.exe`
- `tseditor.prj` creates `tseditor.exe`

Test Executive User Interface Files

The following user interface (`.uir`) files must be located in the same directory as the Test Executive executable:

- `txloginu.uir`
- `txgui.uir`
- `txrtvwu.uir`
- `txengine\txdbuir.uir`
- `txengine\txedsequ.uir`

DLL Modules

Table 6-1 tells you when to include special add-on DLLs to your Test Executive distribution kit and also tells the location of each DLL file. You should distribute the DLLs in the same directory as the Test Executive executable.

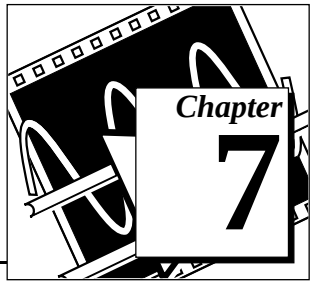
Table 6-1. DLLs for a Test Executive Distribution Kit

Name	Location of DLL	When the DLL is Required
CallExe	`callexe\txshar16.dll` (Windows 3.1) `callexe\txshar32.dll` (Windows 95/NT)	When the Test Executive executable contains the `txcallex.c` module
CallVI	`callvi\callvi32.dll`	When the Test Executive executable contains the `txcallvi.c` module (Windows 95/NT only)
CallVB	`callvb\txcallvb.dll`	When the Test Executive executable contains or loads the `txcallvb.lib` module (Windows 95/NT only)

Files for Database Connectivity

The `tstsuite.prj` and `tseditor.prj` project files use the LabWindows/CVI SQL Toolkit for database connectivity. For more information on distributing the SQL toolkit support files, see the section *Distribution of an SQL Toolkit Executable or DLL* in the `readme.txt` file for the LabWindows/CVI SQL Toolkit.

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 rests on a foundation 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*.

C programmers can easily modify the Test Executive by editing the source code, but not the underlying engine (the function library). If you need to make fundamental changes to the Test Executive itself, you can develop your own test executive source code 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 unless you are an experienced C programmer with thorough knowledge of test executive systems.

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 7-1. Test Executive Engine Function Tree

Test Executive Engine	
Engine	
Open Engine	***TX_OpenEngine***
Close Engine	***TX_CloseEngine***
Get Engine Attribute	***TX_GetEngineAttribute***
Set Engine Attribute	***TX_SetEngineAttribute***
Sequences	
Open Sequence	***TX_OpenSequence***
Close Sequence	***TX_CloseSequence***
Save Sequence	***TX_SaveSequence***
Save Copy of Sequence	***TX_SaveSequenceCopy***
New Sequence	***TX_NewSequence***
Is Sequence Loaded	***TX_IsSequenceLoaded***
Get Sequence Attribute	***TX_GetSeqAttribute***
Set Sequence Attribute	***TX_SetSeqAttribute***
Get Test Preconditions	***TX_GetTestPreconditions***
Low Level	
Get Number of Loaded Sequences	***TX_GetNumSeqIds***
Get Nth Sequence ID	***TX_GetNthSeqId***
Load Sequence	***TX_LoadSequence***
Load Test	***TX_LoadTest***
Load Sequence Pre/Posttest	***TX_LoadSeqPrePostTest***
Release Sequence	***TX_ReleaseSequence***
Release Test	***TX_ReleaseTest***
Release Sequence Pre/Posttest	***TX_ReleaseSeqPrePostTest***
Unload Sequence	***TX_UnloadSequence***
Save Sequence File	***TX_SaveSequenceFile***
Clear Sequence	***TX_ClearSequence***
Obsolete	
Get Information from Engine	***TX_GetEngineInfo***
Change Settings in Engine	***TX_SetEngineInfo***

Table 7-1. Test Executive Engine Function Tree (Continued)

Test Executive Engine	
Running	
Run Sequence or Test	***TX_RunSeqOrTest***
Run Sequence Change Function	***TX_RunSeqChangeTest***
Set Breakpoint Step Type	***TX_BPSetStepType***
Set Current Test	***TX_SetCurrTest***
Low Level	
Run Pretest (Setup Function)	***TX_RunPreTest***
Run Posttest (Cleanup Function)	***TX_RunPostTest***
Run Test	***TX_RunTest***
Run Sequence Pre/Posttest	***TX_RunSeqPrePostTest***
Database	
View Sequence Database	***TX_DBSeqResultsBrowser***
Create Sequence Databases	***TX_DBCreateTables***
Result Handling	
Get Next Result ID	***TX_GetNextResultId***
Get the Number of Results	***TX_GetNumResults***
Get Result Attribute	***TX_GetResultAttribute***
Set Result Attribute	***TX_SetResultAttribute***
Editing	
Edit Sequence	***TX_RunEditSequence***
Memory Management	
Allocate Memory	***TX_Malloc***
Deallocate Memory	***TX_Free***
Reallocate Memory	***TX_Realloc***
Duplicate String	***TX_StrDup***

Table 7-1. Test Executive Engine Function Tree (Continued)

Test Executive Engine	
Utility	
Filename Manipulation	
Get Base Filename	***TX_GetBaseFilename***
Get File Directory	***TX_GetFileDir***
Get File Extension	***TX_GetFileExt***
Make Short Filename	***TX_MakeShortFileName***
Menu Lists	
Create Menu List	***TX_CreateMenuList***
Delete Menu List	***TX_DeleteMenuList***
Delete Menu List Item	***TX_DeleteMenuListItem***
Add Item to Menu List	***TX_AddMenuItemToList***
Get Num Menu List Items	***TX_GetNumMenuListItems***
Get Menu List Attribute	***TX_GetMenuListAttribute***
Set Menu List Attribute	***TX_SetMenuListAttribute***
Get File List from INIFILE	***TX_GetFileListFromIniFile***
Put File List in INIFILE	***TX_PutFileListInIniFile***
INIFILE extensions	
Read Registry into INIFILE	***TX_ReadRegistryInfo***
Write Registry from INIFILE	***TX_WriteRegistryInfo***
Generate Compare String	***TX_GenerateCompareTypeString***
Error Handling	
Get Error String	***TX_GetErrorString***

The headings in bold text in the left column 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 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.

- **Engine** functions open and close the Test Executive Engine as well as getting and setting engine attributes.
- **Sequences** functions opening, closing and saving sequences and getting information about sequences and the tests within sequences.
- **Running** functions run sequences and tests and handle breakpoints.
- **Database** functions access database test results.
- **Result Handling** functions access previous test results.

- **Editing** functions invoke the sequence editor.
- **Memory Management** functions replace the standard *malloc*, *free*, and *realloc* functions. By using the Memory Management functions, you ensure that the Test Executive and the tests use the same set of functions to manage memory.
- **Utility** functions provide utility routines.
- **Error Handling** functions provide access to error messages.

Include Files

The include file, `txengine.h`, of the Test Executive Engine contains function declarations and defined constants for all of the library routines. This include file must be a part of 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. Negative error codes indicate an error. Positive error codes indicate a warning. Zero indicates the function completed successfully. See Appendix A, *Error Codes and Attribute Constants*, for a list of error codes.

Test Executive Engine Function Reference

This section presents descriptions of each function in the Test Executive Engine, in alphabetical order.

TX_AddMenuItemToList

`void` **TX_AddMenuItemToList** (`menuList` **menuListHandle**, `int` **insertPosition**
`char *` **menuItemName**, `void *`**callbackData**);

Purpose

Add a menu item to a menu list.

☞ **Note:** ***You use the*** `TX_CreateMenuList` ***function to set the maximum number of items that a menu list can have. If you add a new item to the bottom of a list that already has the maximum number of items, the function ignores the new item.***

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	menuList	The specifier used to reference the menu list. This is the handle returned by `TX_CreateMenuList`.
	insertPosition	integer	The position at which to insert the menu item. The position may be a number from 1 to the number of menu items in the list, `FRONT_OF_LIST`, or `END_OF_LIST`. The items from the specified position to the end of the list are moved up one position.
	menuItemName	string	The name of the new menu item. To define one of the letters in the name as a shortcut key, place two underscores before this letter.
	callbackData	pointer to void	NULL, or optional pointer to your callback data. The callback data is passed to the menu list callback function.

TX_AddMenuItemToList

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_BPSetStepType

`void` **TX_BPSetStepType** (`int` **stepType**);

Purpose

Set the step type for breakpoints. The step type determines the next breakpoint other than breakpoints defined by the user. The value of step type can be continue, step into, step over, or finish current sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**stepType**	integer	The step type. Valid values: `TXBP_NO_STEP`—Continue until next user breakpoint. `TXBP_STEP_INTO`—Execute the next test, but step into subsequences. `TXBP_STEP_OVER`—Execute the next test, but step over subsequences. `TXBP_FINISH_SEQ`—Finish the current sequence or subsequence.

TX_ClearSequence

`int` **status** = **TX_ClearSequence** (`int` **sequenceId**);

Purpose

Clear the specified sequence. This function frees the memory that the sequence uses, but does not unload the sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence to clear.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Refer to Appendix A for error codes.

TX_CloseEngine

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

Purpose

Close the Test Executive engine. Your program should call this function, at the end of Test Executive execution.

☞ **Note:** ***While*** `TX_CloseEngine` ***closes any remaining open sequences, it is important that you close any open sequence with*** `TX_CloseSequence` ***before calling*** `TX_CloseEngine`***.***

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_CloseSequence

`int` **status** = **TX_CloseSequence** (`int` **sequenceId**, `short` **unloadMode**,
`short` **skipCleanupTest**);

Purpose

Close an open sequence. This function closes only one instance of a sequence. When the sequence file is opened multiple times as a subsequence, the other instances remain open.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence to close.
	unloadMode	short integer	The unload mode of the sequence. Valid values: `USE_SEQ_LOAD_SPEC`—Use the load specification for the sequence to determine which tests to unload. `STATIC_LOAD`—Unload all tests in the sequence. `DYNAMIC_LOAD`—Do not unload any tests. `USE_TEST_LOAD_SPEC`—Use the load specification of each test to determine whether to unload the test.
	skipCleanupTest	short integer	Determines whether the unload cleanup function is skipped.

TX_CloseSequence

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_CreateMenuList

`int` **menuList** = **TX_CreateMenuList** (`int` **menuBarHandle**, `int` **menuID**,
`int` **beforeMenuItemID**, `int` **maxItems**,
`menuListCallbackPtr` **callbackFunction**);

Purpose

Create a new menu list to reside on the specified destination menu bar and returns the new Menu List handle. Subsequent function calls use this new Menu List handle to specify this menu list.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuBarHandle**	integer	The specifier used to reference the menu bar. This is the handle that `LoadMenuBar`, `GetPanelMenuBar`, `NewMenuBar` or `GetPanelMenuBar` return if the menu bar was automatically loaded through `LoadPanel`.
	menuID	integer	The ID number for a particular menu within a menu bar. The Menu ID is a constant name (located in the UIR header file) that the LabWindows/CVI User Interface Editor generates or a value returned by the `NewMenu` function.
	beforeMenuItemID	integer	The new menu item appears above this existing menu item. Pass `-1` to place the new item at the end (bottom) of the menu list.
	maxItems	integer	Specifies the maximum number of items in the list. **Note:** If you add a new item to the bottom of a list that already has the maximum number of items, the new item is ignored.

TX_CreateMenuList

(Continued)

Input/ Output	Parameter Name	Data Type	Purpose
	callbackFunction	MenuList CallbackPtr	Callback function for the menu list item or NULL if a callback function is not needed.

Return Value

Variable Name	Data Type	Meaning
menuList	integer	Handle that specifies this menu list in subsequent function calls. Negative values indicate that an error occurred.

Parameter Discussion

The callback function adheres to the following prototype:

```
void CVICALLBACK Callback_Function(menuList list, int menuIndex,
                                   int event, void *callbackData);
```

The callback event function is passed the menu list handle, menu list index, the event, and the callback data. Callback data for each menu item is specified when each menu item is added. See `TX_AddMenuItemToList`.

The following two callback events are possible:

- `EVENT_COMMIT`, when the user selects a menu list item
- `EVENT_DISCARD`, when the program discards a menu list item

TX_DBCreateTables

`int` **status** = **TX_DBCreateTables** (`int` **sequenceId**);

Purpose

Create new tables for the database associated with the specified sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence for which the function creates database results tables.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_DBSeqResultsBrowser

`int` **menuList** = **TX_DBSeqResultsBrowser** (`int` **sequenceId**);

Purpose

Display the database associated with the specified sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence that created the results.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_DeleteMenuList

`int` **status** = **TX_DeleteMenuList** (`menuList` **menuListHandle**);

Purpose

Delete a menu list from a menu bar.

This function calls the menu list callback function (if defined) and passes the `EVENT_DISCARD` event for each menu list item in the menu list.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	integer	The specifier used to reference the menu list. This is the handle returned by `TX_CreateMenuList`.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_DeleteMenuListItem

`int` **status** = **TX_DeleteMenuListItem** (`menuList` **menuListHandle**, `int` **item**);

Purpose

Delete a menu list item from a menu list.

This function calls the menu list callback function (if defined) and passes the `EVENT_DISCARD` event for the deleted item.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menulisthandle**	integer	The specifier used to refer to the menu list. This is the handle returned by `TX_CreateMenuList`.
	item	integer	The position of the item to delete. Valid values: `1` to the number of items in the list.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_Free

`int` **TX_Free** (`void *` **memBlockPointer**);

Purpose

Causes deallocation of the memory that **memBlockPointer** points to. In other words, the region of memory becomes unavailable for further use.

Use `TX_Free` instead of the ANSI C `free` function whenever you want to free memory returned by a test or to free memory allocated with `TX_Malloc`. The following functions use `TX_Malloc`: `TX_StrDup`, `TX_GetEngineAttribute`, `TX_GetSeqAttribute`, and `TX_GetResultAttribute`. For tests, `data->freeFuncPtr` points to `TX_Free`.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**memBlockPointer**	pointer	Pointer to the memory block to free.

TX_GenerateCompareTypeString

`int` **status** = **TX_GenerateCompareTypeString** (`int` **comparisonType**, `double` **limit1**, `double` **limit2**, `char *` **comparisonString**);

Purpose

Generate a string description of a comparison specification, as in the following example:

```
{GELT(>= && <), 4.0, 5.0}
```

Parameters

Input/ Output	Parameter Name	Data Type	Purpose

Input	**comparisonType**	integer	The type of comparison. Valid values:
			`SPEC_EQ` `SPEC_GELE`
			`SPEC_NEQ` `SPEC_GTLE`
			`SPEC_GT` `SPEC_GELT`
			`SPEC_GE` `SPEC_BOOL`
			`SPEC_LT` `SPEC_LOG`
			`SPEC_LE` `SPEC_NONE`
			`SPEC_GTLT`

	limit1	double	The first limit value. The function ignores this value if the comparison type does not test against a returned value (Boolean, Log, or None limit specifications).
	limit2	double	The second limit value. The function ignores this value if the comparison type does not test against a returned value or if the comparison type uses only one limit.
Output	**comparisonString**	string	A string description of the comparison specification.

TX_GenerateCompareTypeString

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_GetBaseFilename

`char *` **baseFilename** = **TX_GetBaseFileName** (`char *` **fullPathFilename**);

Purpose

Get a filename, given a full pathname that includes the filename.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**fullPathFilename**	string	The full pathname (including the filename and extension).

Return Value

Variable Name	Data Type	Meaning
baseFilename	string	Copy of the filename, including the extension. This function uses `TX_Malloc`; use `TX_Free` to free the returned value.

TX_GetEngineAttribute

`int` **status** = **TX_GetEngineAttribute** (`int` **attribute**, `void * ` **value**);

Purpose

Get information from the Test Executive engine.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**attribute**	integer	Indicates what information to get. See Appendix A for a complete list of engine attributes.
Output	**value**	pointer to void	The value of the specified attribute. See Appendix A for a complete list of engine attributes.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

Some attributes specify the system callback function. The prototype for a system callback function is:

`int` **UserCallback** (`int` **callbackEvent**, `int` **seqId**, `int` **runTestType**, `int` **resultId**);

For string attributes, this function copies the attribute value to the memory specified by this parameter. The use of the `TX_Free` function is not required.

TX_GetEngineInfo

`int` **status** = **TX_GetEngineInfo** (`unsigned short` **attribute**, `void *` **value**);

Purpose

This function is superseded by `TX_GetSeqAttribute`.

`TX_GetEngineInfo` returns the information stored in the Engine for the current sequence and/or test. The data it returns is only a copy of the actual data in the engine. Changes to the return value do not affect the data inside the engine.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**attribute**	unsigned short integer	Indicates what information to get.
Output	**value**	pointer to void	Pointer to hold the returned information.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_GetErrorString

`char *` **errorMessage** = **TX_GetErrorString** (`int` **errorCode**);

Purpose

Return the error message text associated with an error code.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**errorCode**	integer	The error code returned by a Test Executive function. Appendix A lists all error and warning codes.

Return Value

Variable Name	Data Type	Meaning
errorMessage	string	The text error message associated with the error code.

TX_GetFileDir

`char * filePath = TX_GetFileDir (char * fullPathFilename);`

Purpose

Get a directory pathname, given a pathname that includes the filename.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**fullPathFilename**	string	The full pathname (including the filename and extension).

Return Value

Variable Name	Data Type	Meaning
filePath	string	Copy of the directory pathname, excluding the filename. This function uses `TX_Malloc`; use `TX_Free` to free the returned value.

TX_GetFileExt

`char *` **fileExtension** = **TX_GetFileExt** (`char *` **fullPathFilename**);

Purpose

Return a pointer to the file extension portion of a filename.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**fullPathFilename**	string	The full pathname (including the filename and extension).

Return Value

Variable Name	Data Type	Meaning
fileExtension	string	Pointer to the file extension. The calling function should not free this value.

TX_GetFileListFromIniFile

```
int status = TX_GetFileListFromIniFile (menuList menuListHandle,
                                        IniText INIFILEHandle,
                                        const char * sectionName,
                                        const char * tagPrefix, int flags);
```

Purpose

Get a filename list from an INIFILE instrument driver handle and place the list in a menu list.

The filename list must have been placed into the INIFILE using the `TX_PutFileListInIniFile` function.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	menuList	The specifier used to reference the menu list. This is the handle returned by `TX_CreateMenuList`.
	INIFILEHandle	IniText	The handle returned from `Ini_New` in the INIFILE instrument driver. It specifies the list of in-memory tag/value pairs.
	sectionName	string	The section name containing the tag/value pairs.
	tagPrefix	string	The tag name prefix used in the tag/value pairs.

TX_GetFileListFromIniFile

(Continued)

Input/ Output	Parameter Name	Data Type	Purpose
	flags	integer	Specifies special flags when creating menu list. Valid values: `0`: No special flags. `1`: Use `TX_MakeShortFileName` to create a short filename for the visible menu item and use `TX_Malloc` to create a long filename as the callback data. **Note:** You must use `TX_Free` to free the callback data before deleting the menu list. For example, you can free the callback data on the event `EVENT_DISCARD` in the menu list callback function.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_GetMenuListAttribute

`int` **status** = **TX_GetMenuListAttribute** (`menuList` **menuListHandle**, `int` **item**,
`int` **attribute**, `void *` **value**);

Purpose

Obtain the value of an attribute for a menu list or menu list item.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	menuList	The specifier used to reference the menu list. This is the handle that `TX_CreateMenuList` returns.
	item	integer	The position of the menu item. The position may be a number from `1` to the number of items in the list.
	attribute	integer	Menu List attributes. See Appendix A for a list of menu attributes.
Output	**value**	void *	The value of the attribute.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_GetNextResultId

`void` **TX_GetNextResultId** (`int` **startingResultId**, `int` **stepInto**, `int *` **nextResultId**);

Purpose

Get the result ID of the next test or sequence result.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**startingResultId**	integer	The starting result ID. If the value is `-1`, start at the current result ID.
	stepInto	integer	Determines how subsequences are handled; whether to get the results in subsequences.
Output	**nextResultId**	pointer to integer	The result ID number of the next test or sequence result.

TX_GetNthSeqId

`int` **sequenceId** = **TX_GetNthSeqId** (`int` **NthSequence**);

Purpose

Return the sequence ID number of the *n*th sequence currently loaded.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**NthSequence**	integer	The position of the item to be retrieved. The position may be a number from `1` to the number of sequences loaded.

Return Value

Variable Name	Data Type	Meaning
sequenceId	integer	The Sequence ID number of the *n*th sequence.

TX_GetNumMenuListItems

`int` **numItems** = **TX_GetNumMenuListItems** (`menuList` **menuListHandle**);

Purpose

Return the number of menu items in a menu list.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	integer	The specifier used to reference the menu list. This is the handle that `TX_CreateMenuList` returns.

Return Value

Variable Name	Data Type	Meaning
numItems	integer	The number of items in the menu list or zero if an error occurs.

TX_GetNumResults

`void` **TX_GetNumResults** (`int` **startingResultId**, `int` **stepInto**,
`int *` **numberOfResults**);

Purpose

Count the number of results in a result tree or subtree. You can count the results in only the starting sequence or the sequence and its subsequences.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**startingResultId**	integer	The starting result ID. If the value is `-1`, start at the current result ID.
	stepInto	integer	Determines how subsequences are handled; whether to count the results in subsequences.
Output	**numberOfResults**	pointer to integer	The number or results in the result tree or subtree.

TX_GetNumSeqIds

`int` **numberOfSequences** = **TX_GetNumSeqIds** (`void`);

Purpose

Return the number of loaded sequences.

Return Value

Variable Name	Data Type	Meaning
numberOfSequences	integer	The number of sequence IDs, in other words, the number of unique sequences loaded.

TX_GetResultAttribute

`int` **status** = **TX_GetResultAttribute** (`int` **resultId**, `int` **attribute**, `void *` **value**);

Purpose

Get information about the result of a test or sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**resultId**	integer	The result ID for the test or sequence result.
	attribute	integer	The attribute to get. See Appendix A for a complete list of result attributes.
Output	**value**	pointer to void	Pointer to hold the returned information.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

This function uses `TX_Malloc` for string return values. Use `TX_Free` to free the returned value.

TX_GetSeqAttribute

`int` **status** = **TX_GetSeqAttribute** (`int` **sequenceId**, `int` **testNumber**, `int` **attribute**, `void *` **value**);

Purpose

Get information about a sequence or a test within a sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence from which to get the information. If the value is `-1`, uses the currently executing sequence ID.
	testNumber	integer	The test number for which information is desired. If the value is `-1`, the current test is used. If the attribute does not apply to tests, this parameter is ignored.
	attribute	integer	The attribute to get. See Appendix A for a complete list of sequence and test attributes.
Output	**value**	pointer to void	Pointer to hold the returned information. See *Parameter Discussion*.

Parameter Discussion

For attributes that return string, array of string, or any pointer to memory, the returned memory is dynamically allocated with `TX_Malloc`. Use `TX_Free` to deallocate the returned memory.

TX_GetSeqAttribute

(Continued)

Consider the following example:

```
int i;
char *testInfo[8] = {0};
TX_GetSeqAttribute (seqId, -1, TXATTR_TEST_NAMES, testInfo);

/* Use the values here */

/* Free the values */
for (i=0;i<8;i++)
      TX_Free(testInfo[i]);
```

The data this function returns is only a copy of the actual data in the engine. Changes to the return value do not affect the data inside the engine.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_GetTestPreconditions

```
char * precondStr = TX_GetTestPreconditions (int sequenceId, int testIndex
                                             char * prefixBuffer);
```

Purpose

Return a string describing the preconditions of a test.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence.
	testIndex	integer	The number of the test for which precondition information is to be returned.
	prefixBuffer	string	Optional prefix to attach to the description of the preconditions of the test.

Return Value

Variable Name	Data Type	Meaning
precondStr	integer	A string containing the description of the preconditions of the specified test.

TX_IsSequenceLoaded

`int` **isLoaded** = **TX_IsSequenceLoaded** (`int` **searchByFilename**, `int *` **sequenceId**, `char *` **sequenceFile**);

Purpose

Returns data regarding whether a specified sequence is currently loaded.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**searchByFilename**	integer	Tells whether to look for the sequence by filename or sequence ID.
Input/ Output	**sequenceId**	integer	Pointer to the sequence ID to search for. Can also return the sequence ID when the user finds a sequence file through a filename search.
	sequenceFile	string	Pointer to the full pathname (including filename and extension) to search for. Can also return the sequence ID when the user finds a sequence file through a filename search.

Return Value

Variable Name	Data Type	Meaning
isLoaded	integer	Indicates whether the sequence is currently loaded.

TX_LoadSeqPrePostTest

`int` **status** = **TX_LoadSeqPrePostTest** (`int` **sequenceId**, `int *` **isLoadUnloadTest**, `int` **isSetupTest**);

Purpose

Load a sequence setup/cleanup function or a load/unload function.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the setup/cleanup function.
	isLoadUnloadTest	integer	Indicates whether the function to be loaded is a sequence load/unload function.
	isSetupTest	integer	Indicates whether the function is a setup test rather than a cleanup test.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_LoadSequence

`int` **status** = **TX_LoadSequence** (`char *` **filename**, `int *` **sequenceId**);

Purpose

Load a sequence into the engine. This function does not load the tests or subsequences that the sequence uses.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**filename**	string	The full pathname (including filename and extension) of the sequence to load.
Output	**sequenceId**	integer	The returned sequence ID number of the loaded sequence.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_LoadTest

`int` **status** = **TX_LoadTest** (`int` **sequenceId**, `int` **testNumber**);

Purpose

Load a single test and its setup and cleanup tests.

☞ **Note:** ***If*** `TX_LoadTest` ***is executed on a subsequence test, this function only loads the setup and cleanup tests. It does not load the subsequence.***

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the test.
	testNumber	integer	The number of the test to load.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_MakeShortFileName

```
char * returnedString = TX_MakeShortFileName (char * shortName,
                                              char * longName, int maxSize);
```

Purpose

Generate a short filename string (including the terminating ASCII NUL byte) from a target filename string.

Consider the following function call:

```
TX_MakeShortFileName(buffer, "c:\\cvi\\samples\\file.c", 20);
```

The preceding function call generates the following output string:

```
"c:\...\samples\file.c"
```

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Output	**shortName**	string	The target string to which the generated shortened filename string is copied (including the terminating ASCII NUL byte). If a NULL is passed, an internal buffer will be used.
Input	**longName**	string	Pointer to the NUL-terminated filename string that is the source of the shortened filename.
	maxSize	integer	The maximum number of characters allowed in the output string.

Return Value

Variable Name	Data Type	Meaning
returnedString	string	Pointer to the generated string. If **shortName** is NULL, the function returns an address to an internal static buffer.

TX_Malloc

`void *` **memBlock** = **TX_Malloc** (`size_t` **memBlockSize**);

Purpose

Allocate space for an object of specified size. The space you allocate has indeterminate contents.

Use `TX_Malloc` instead of `malloc` whenever you want to allocate memory that a test will use or that you will free with `TX_Free`.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**memBlockSize**	size_t	The size in bytes of the space you are allocating.

Return Value

Variable Name	Data Type	Meaning
memBlock	pointer to void	A pointer to the memory block allocated. If the function cannot allocate the space or if the size of the space requested is zero, the function returns a NULL pointer.

TX_NewSequence

`int` **sequenceId** = **TX_NewSequence** (`void`);

Purpose

Create a new, empty sequence and return a sequence ID number for the new sequence.

Return Value

Variable Name	Data Type	Meaning
sequenceId	integer	The sequence ID number for the new sequence. If the value is `-1`, the function was unable to create a new sequence.

TX_OpenEngine

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

Purpose

Initialize the Test Executive Engine. You must call `TX_OpenEngine` at the beginning of Test Executive execution.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_OpenSequence

`int` **status** = **TX_OpenSequence** (`char *` **sequenceFile**, `short` **loadMode**,
`short` **skipSetupTest**, `int *` **openedSequenceId**);

Purpose

Open a sequence for use by the Test Executive.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceFile**	integer	Full pathname (including filename and extension) of sequence. If the filename is the empty string (in other words, `""`) the function prompts the user for a filename.
	loadMode	short integer	The loading mode of the sequence. Valid values: `STATIC_LOAD`—Load all tests and subsequences when the sequence is loaded. `DYNAMIC_LOAD`—Load tests and subsequences as needed. `USE_TEST_LOAD_SPECS`—Use the load specification of each test to determine whether to load the test statically or dynamically.
	skipSetupTest	short integer	Determines whether to skip the load setup function.
Output	**openedSequenceId**	integer	The Sequence ID number of the opened sequence. The Sequence ID is a parameter for many Test Executive functions. If the value is `-1`, the function was unable to open the sequence.

TX_OpenSequence

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_PutFileListInIniFile

```
int status = TX_PutFileListInIniFile (menuList menuListHandle,
                                      IniText INIFILEHandle,
                                      const char * sectionName,
                                      const char * tagPrefix, int baseTagNameToUse);
```

Purpose

Put a menu list to the handle of an INIFILE instrument driver. You can use the `TX_GetFileListFromIniFile` function to retrieve the menu list.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	menuList	The specifier used to reference the menu list. This is the handle that `TX_CreateMenuList` returns.
	INIFILEHandle	IniText	The handle returned from `Ini_New` in the INIFILE instrument driver. It represents the list of in-memory tag/value pairs.
	sectionName	string	The section name under which to place the tag/value pairs.
	tagPrefix	string	The tag name prefix to use in the tag/value pairs.

TX_PutFileListInIniFile

(Continued)

Input/ Output	Parameter Name	Data Type	Purpose
	baseTagNameToUse	int	Specify the base tag name to be placed into INIFILE. Values: `0`: Use menu item name as base tag name. `1`: Use each menu item's callback data as base tag name. This assumes that the callback data is a NUL-terminated string. This option is useful when the menu item name is a short filename (see `TX_MakeShortFileName`) and the callback data is the long filename, as in the following example: menu item = `"c:\...\file.c"`

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_ReadRegistryInfo

`int` **status** = **TX_ReadRegistryInfo** (`IniText` **INIFILEHandle**,
`const char *` **registryName**);

Purpose

Create an INIFILE handle and read tag/value pairs from either the Windows Registry or a file.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**INIFILEHandle**	IniText	The handle returned from `Ini_New` in the INIFILE instrument driver. It represents the list of in-memory tag/value pairs.
	registryName	string	Specifies the location of the tag/value pairs. Windows 95/NT—Specifies the Windows Registry location from which to read the INIFILE information. Windows 3.1/UNIX—Specifies the filename from which to read the INIFILE information.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_Realloc

`void *` **memBlock** = **TX_Realloc** (`void *` **memBlockPtr**, `size_t` **newSize**);

Purpose

Change the size of a memory block that you have previously allocated and preserves the contents. If the new size is larger, the contents of the newly allocated portion of the object is indeterminate. If `TX_Realloc` cannot allocate the space, the memory block remains unchanged.

Use `TX_Realloc` instead of `realloc` to reallocate memory originally allocated with `TX_Malloc`. You must use `TX_Free` to free memory reallocated by `TX_Realloc`.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**memBlockPtr**	pointer to void	A pointer to the memory block where space reallocation occurs. If this control contains a NULL pointer, this function behaves like `TX_Malloc`. If the pointer does not match one you obtained earlier by `TX_Malloc`, or `TX_Realloc`, the behavior is undefined.
Input	**newSize**	size_t	The new size for the memory block. If the new size is larger, the contents of the newly allocated portion of the memory block is indeterminate. If the new size is smaller, the contents at the end of the block are lost. If the size is zero and if **memBlockPtr** is not a null pointer, the object it points to is freed.

TX_Realloc

(Continued)

Return Value

Variable Name	Data Type	Meaning
memBlock	pointer to void	Pointer to the moved allocated space. If the space cannot be allocated or if the size of the space requested is zero, the call returns a null pointer.

TX_ReleaseSeqPrePostTest

`int` **status** = **TX_ReleaseSeqPrePostTest** (`int` **sequenceId**, `int` **isLoadUnloadTest**, `int` **isSetupTest**);

Purpose

Release a sequence setup/cleanup function or a load/unload function.

Parameters

Input	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the setup/cleanup function.
	isLoadUnloadTest	integer	Indicates whether the function to be released is a sequence load/unload function.
	isSetupTest	integer	Indicates whether the function is a setup test rather than a cleanup test.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_ReleaseSequence

`int` **status** = **TX_ReleaseSequence** (`int` **sequenceId**);

Purpose

Decrease the reference count for a sequence that a program has loaded using `TX_LoadSequence`.

Whenever a program successfully calls `TX_LoadSequence` on a sequence, the reference count of the sequence increments by one. Whenever a program calls `TX_ReleaseSequence`, the reference count of the sequence decrements by one. If the reference count decrements to zero, the engine unloads the sequence and invalidates the sequence ID.

If you want to unload the sequence regardless of the reference count, call `TX_UnloadSequence`.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence to release.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_ReleaseTest

`int` **status** = **TX_ReleaseTest** (`int` **sequenceId**, `int` **testNumber**);

Purpose

Decrease the reference count for a test that a program has loaded using `TX_LoadTest`.

Whenever a program successfully calls `TX_LoadTest` on a test, the reference count of the test increments by one. Whenever a program calls `TX_ReleaseTest`, the reference count of the test decrements by one. If the reference count decrements to zero, the test is unloaded.

If you want to unload the test regardless of the reference count, call `TX_UnloadTest`.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the test to be released.
	testNumber	integer	The number of the test to be released. If the value is `-1`, the function releases the current test.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_RunEditSequence

`int` **status** = **TX_RunEditSequence** (`int *` **sequenceId**, `int` **loadAndDiscardPanel**, `int` **isStandAlone**);

Purpose

Invoke the Sequence Editor user interface.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	Pointer to the sequence ID number of the sequence to edit. To edit a new sequence, set the parameter to -1.
	loadAndDiscardPanel	integer	Indicates whether to load the editor user interface panels before editing and unload them afterwards.
	isStandAlone	integer	Indicates whether the sequence editor is being used as a standalone utility.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_RunPostTest

`int` **status** = **TX_RunPostTest** (`int` **sequenceId**, `void *` **testData**, `void *` **testError,**
`int *` **postTestError**);

Purpose

Run the posttest (cleanup function) for the current individual test or subsequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the posttest.
Input/ Output	**testData**	tTestData	Pointer to structure containing the test data.
	testError	tTestError	Pointer to structure returning error information.
Output	**posttestError**	integer	Flag indicating an error occurred in the posttest function.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

Sequence pretests and posttests usually ignore the **testData** and **testError** structures.

TX_RunPreTest

`int` **status** = **TX_RunPreTest** (`int` **sequenceId**, `void *` **testData**, `void *` **testError**, `int *` **preTestError**);

Purpose

Run the pretest (setup function) for the current individual test or subsequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the pretest.
Input/ Output	**testData**	tTestData	Pointer to structure containing the test data.
	testError	tTestError	Pointer to structure returning error information.
Output	**pretestError**	integer	Flag indicating an error occurred in the posttest function.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

Sequence pretests and posttests usually ignore the **testData** and **testError** structures.

TX_RunSeqChangeTest

`int` **status** = **TX_RunSeqChangeTest** (`int` **sequenceId**, `int` **functionType**,
`int` **loopCount**, `int *` **result**);

Purpose

Run the setup or cleanup function of a sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the test function.
	functionType	integer	Tells whether to run the open sequence function or the close sequence function. Valid values: `TXRT_OPEN_SEQ`—load setup function. `TXRT_CLOSE_SEQ`—unload cleanup function. `TXRT_PRE_SEQ`—sequence setup function. `TXRT_POST_SEQ`—sequence cleanup function.
	loopCount	integer	Number of times to run the function. Usually, this value is 1.
Output	**result**	integer	Returns the PASS/FAIL result of test. If the program passes a NULL, this parameter is ignored. Possible values are `FAIL`, `PASS`, `SKIP`, and `ABORT`.

TX_RunSeqChangeTest

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_RunSeqOrTest

`int` **status** = **TX_RunSeqOrTest** (`int` **sequenceId**, `int` **whatToRun**, `int` **testNumber**, `int` **loopCount**, `int *` **result**);

Purpose

Run a sequence or test.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence.
	whatToRun	integer	Tells what to run. Valid values: `TXRT_SINGLE_TEST`—run a single test once. `TXRT_LOOP_TEST`—run a single test multiple times. `TXRT_SINGLE_PASS`—run a sequence once (typically without a UUT number). `TXRT_LOOP_SEQ`—run a sequence multiple times. (typically a UUT number is used).
	testNumber	integer	The test number of the test to run. If the test type is `TXRT_SINGLE_PASS` or `TXRT_LOOP_SEQ`, the program ignores this parameter.
	loopCount	integer	The number of times to loop. For `TXRT_LOOP_TEST` or `TXRT_LOOP_SEQ`. If the value is `-1`, loop until abort or fail.
Output	**result**	integer	Returns the PASS/FAIL result of sequence or test. If a NULL is passed, this parameter is ignored. Possible values are `FAIL`, `PASS`, `SKIP`, and `ABORT`.

TX_RunSeqOrTest

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_RunSeqPrePostTest

```
int status = TX_RunSeqPrePostTest (int sequenceId, void * testData,
                                   void * testError, int isLoadUnloadTest,
                                   int isSetupTest);
```

Purpose

Run a sequence setup/cleanup function or a load/unload function.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the pretest or posttest.
Input/ Output	**testData**	tTestData	Pointer to structure containing the test data.
	testError	tTestError	Pointer to structure returning error information.
Input	**isLoadUnloadTest**	integer	Tells whether the function is a sequence load/unload function.
	isSetupTest	integer	Tells whether the function is a setup test rather than a cleanup test.

TX_RunSeqPrePostTest

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

Sequence pretests and posttests usually ignore the **testData** and **testError** structures.

TX_RunTest

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

Purpose

Run an individual test function and its setup and cleanup functions, if any.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the test to run.
Input/ Output	**testData**	tTestData	Pointer to structure containing the test data. This pointer will be passed to the test function.
	testError	tTestError	Pointer to a structure for returning error information from the test function.
Output	**prePostTestError**	pointer to integer	Flag indicating an error occurred in the pretest or posttest.

TX_RunTest

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

Pretests and posttests usually ignore the `testData` and `testError` structures. You have the option to use these structures to share data between the pretest, test, and posttest functions.

TX_SaveSequence

`int` **status** = **TX_SaveSequence** (`int` **sequenceId**, `int` **askBeforeOverwrite**);

Purpose

Save a sequence. If the sequence has not been named, the program prompts the user to name the file.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of an open sequence to be saved.
	askBeforeOverwrite	integer	Indicates whether to prompt the user before overwriting an existing file. **Note:** Whenever the system prompts you to name a file, it always displays another prompt when you choose a name that will overwrite an existing file.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SaveSequenceCopy

`int` **status** = **TX_SaveSequenceCopy** (`int` **sequenceId**, `char *` **filename**,
`short` **askBeforeOverwriting**);

Purpose

Save a copy of a sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of an open sequence to be saved.
	filename	string	The full pathname (including filename and extension) to use as the filename for the saved sequence. If **filename** is NULL or the empty string (""), the function prompts the user to name the file.
	askBeforeOverwriting	short integer	Indicates whether to prompt the user before overwriting an existing file.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SaveSequenceFile

`int` **status** = **TX_SaveSequenceFile** (`char * `**filename**, `int` **sequenceId**);

Purpose

Write a sequence to a specified file.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**filename**	string	The full pathname (including filename and extension) where the sequence will be saved.
	sequenceId	string	The sequence ID number of the sequence to save.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SetCurrTest

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

Purpose

Set the current test in the specified sequence to the specified test number.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence containing the test.
	testNumber	integer	The number of the test which will be the new current test. Test numbering begins with `1`.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SetEngineAttribute

`int` **status** = **TX_SetEngineAttribute** (`int` **attribute**, ...);

Purpose

Set information for the Test Executive engine.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**attribute**	integer	The attribute to set. See Appendix A for a complete list of engine attributes.
	value	pointer to void	The value to which to set the specified attribute.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Parameter Discussion

Some attributes specify system callback function. A system callback function adheres to the following prototype:

`int` **UserCallback** (`int` **callbackEvent**, `int` **seqId**, `int` **runTestType**, `int` **resultId**);

TX_SetEngineInfo

`int` **status** = **TX_SetEngineInfo** (`int` **attribute**, `void * ` **value**);

Purpose

This function is superseded by `TX_SetSeqAttribute`.

`TX_SetEngineInfo` inputs new information into the Test Executive engine for the current sequence or test. The data that the function sends to the engine is copied and saved in the engine. After your program calls this function, any changes to the original data do not affect the data stored in the engine.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**attribute**	integer	The attribute to set.
	value	pointer to void	Pointer containing the new value to be inserted.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SetMenuListAttribute

`int` **status** = **TX_SetMenuListAttribute** (`menuList` **menuListHandle**, `int` **item**, `int` **attribute**, …);

Purpose

Set the value of an attribute for a menu list or menu list item.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**menuListHandle**	menuList	The specifier used to reference the menu list. This is the handle returned by `TX_CreateMenuList`.
	item	integer	The position of the menu item. The position may be a number from 1 to the number of items in the list.
	attribute	integer	Menu List attributes. See Appendix A for a list of menu attributes.
Output	**value**	depends on attribute	The value of the attribute.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SetResultAttribute

`int` **status** = **TX_SetResultAttribute** (`int` **resultId**, `int` **attribute**, ...);

Purpose

Set information about the result of a test or sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**resultId**	integer	The result ID for the test or sequence result.
	attribute	integer	The attribute to set. See Appendix A for a complete list of result attributes.
Output	**value**	depends on attribute	The value of the specified attribute.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_SetSeqAttribute

`int` **status** = **TX_SetSeqAttribute** (`int` **sequenceId**, `int` **testNumber**, `int` **attribute**, ...);

Purpose

Set attributes of a sequence or a test within a sequence.

The data that the function sends to the engine is copied and stored in the engine. After your program calls this function, any changes to the original data do not affect the data stored in the engine.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence where the attribute will be set. If the value is `-1`, uses the currently executing sequence ID.
	testNumber	integer	The test number for which the attribute will be set. If the value is `-1`, the current test is used. If the attribute does not apply to tests, the system ignores the parameters.
	attribute	integer	The attribute to set. See Appendix A for a complete list of sequence and test attributes.
Output	**value**	depends on attribute	The value(s) of the specified attribute.

TX_SetSeqAttribute

(Continued)

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_StrDup

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

Purpose

Return a duplicate of the input string. You must use `TX_Free` to free the duplicate string.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**stringToDuplicate**	string	String to duplicate.

Return Value

Variable Name	Data Type	Meaning
duplicate	string	Duplicate of input string. If the input string is NULL or memory could not be allocated, this value is NULL.

TX_UnloadSequence

`int` **status** = **TX_UnloadSequence** (`int` **sequenceId**);

Purpose

Unload a sequence; in other words, remove the sequence from memory. You should release all the tests in the sequence with `TX_ReleaseTest` before unloading the sequence.

Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**sequenceId**	integer	The sequence ID number of the sequence to unload.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

TX_WriteRegistryInfo

`int` **status** = **TX_WriteRegistryInfo** (`IniText` **INIFILEHandle**,
`const char *` **registryName**);

Purpose

Write out the tag/value pairs from an INIFILE handle to either the Windows Registry or to a file.

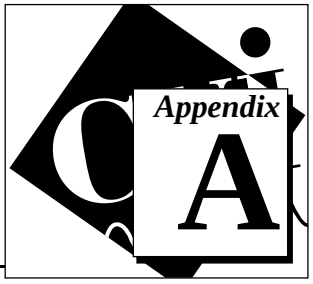
Parameters

Input/ Output	Parameter Name	Data Type	Purpose
Input	**INIFILEHandle**	IniText	The handle that `Ini_New` returns in the INIFILE instrument driver. It represents the list of tag/value pairs in memory.
	registryName	string	Specifies the location of the tag/value pairs. Windows 95/NT—Specifies the Windows Registry location from which to read the INIFILE information. Windows 3.1/Solaris—Specifies the filename from which to read the INIFILE information.

Return Value

Variable Name	Data Type	Meaning
status	integer	0 = Successful Negative Number = Error, function terminated Positive Number = Warning, function completed Appendix A lists all error and warning codes.

Error Codes and Attribute Constants



This appendix lists the error codes and other important constants used by the Test Executive engine.

The header file `txengine.h` contains further information on constants.

Error Codes

The functions in the Test Executive engine can return the error codes listed in Table A-1.

Table A-1. Error Codes

Error Code	Meaning
TXERR_OUT_OF_MEMORY	Cannot allocate memory
TXERR_LIST_EMPTY	Cannot get information, no tests defined
TXERR_ILLEGAL_POINTER	NULL pointer where value required
TXERR_CLIPBOARD_EMPTY	Clipboard empty
TXERR_END_OF_LIST	Unexpected end of test list
TXERR_NO_PRE_POST_TEST	Attempt to execute non-existent pretest or posttest
TXERR_TEST_NOT_VERIFIED	Attempt to run a test which has not been loaded
TXERR_LOCAL_PRETEST_NOT_VERIFIED	Attempt to run a pretest which has not been loaded
TXERR_LOCAL_POSTTEST_NOT_VERIFIED	Attempt to run a posttest which has not been loaded
TXERR_TEST_RUN_TIME	Run-time failure in test
TXERR_TEST_VERIFY_LOAD	Cannot find specified test file
TXERR_PRETEST_VERIFY_LOAD	Cannot find specified pretest file

Table A-1. Error Codes (Continued)

Error Code	Meaning
TXERR_POSTTEST_VERIFY_LOAD	Cannot find specified posttest file
TXERR_TEST_VERIFY_ADDR	Cannot find test in specified file
TXERR_PRETEST_VERIFY_ADDR	Cannot find pretest in specified file
TXERR_POSTTEST_VERIFY_ADDR	Cannot find posttest in specified file
TXERR_READING_FROM_FILE	Error reading from old style binary sequence file
TXERR_NOT_TX_FILE	Sequence file marker missing from binary sequence file
TXERR_WRITING_TO_FILE	Error writing sequence file
TXERR_SEQUENCE_PRETEST_FAIL	Run sequence pretest failed
TXERR_SEQUENCE_POSTTEST_FAIL	Run sequence posttest failed
TXERR_META_PRETEST_FAIL	Load sequence pretest failed
TXERR_META_POSTTEST_FAIL	Unload sequence posttest failed
TXERR_TEST_OUT_OF_RANGE	Specified test number is out of range for sequence
TXERR_INVALID_ENGINE_INFO	Invalid attribute number
TXERR_PRETEST_RUN_TIME	Run-time error in pretest
TXERR_POSTTEST_RUN_TIME	Run-time error in posttest
TXERR_SEQPRETEST_RUN_TIME	Run-time error in run sequence pretest
TXERR_SEQPOSTTEST_RUN_TIME	Run-time error in run sequence posttest
TXERR_METAPRETEST_RUN_TIME	Run-time error in sequence load pretest
TXERR_METAPOSTTEST_RUN_TIME	Run-time error in sequence unload posttest
TXERR_NEWER_VERSION	Sequence file version greater than supported version

Table A-1. Error Codes (Continued)

Error Code	Meaning
TXERR_LIMIT_SPEC_OUT_OF_RANGE	Limit Specification (comparison type) out of range
TXERR_INVALID_TEST_MODULE	Invalid test object file
TXERR_UNABLE_TO_OPEN_FILE	Cannot open file
TXERR_SEQUENCE_NOT_FOUND	Cannot find sequence with specified sequence ID
TXERR_SECTION_MISSING	Sequence file is missing required section
TXERR_MISSING_TEST	Gap between test numbers in sequence file
TXERR_READ_ONLY_VALUE	Can only read value for specified attribute
TXERR_TEST_UNLOAD	Error unloading test
TXERR_SUBSEQ_RECURSION	Subsequence recursion is not supported
TXERR_UNDEF_IN_PRECOND	Undefined test in precondition
TXERR_INVALID_PARAMETER	Value supplied is invalid for the specified attribute
TXERR_INTERNAL_TOOLBOX	Internal error in LabWindows/CVI Toolbox
TXERR_UIR_FILE_NOT_FOUND	User interface file not found
TXERR_SEQUENCE_IS_RUNNING	Operation is illegal when sequence is running
TXERR_ELEMENT_EXISTS	Unused
TXERR_ELEMENT_NOT_FOUND	Unused
TXERR_SM_NOT_CREATED	Unused
TXERR_VALUE_OUT_OF_RANGE	Value out of range

Warning Codes

The functions of the Test Executive engine can return the warning codes listed in Table A-2.

Table A-2. Warning Codes

Warning Code	Meaning
TXWNG_LOAD_FAILURE	Failure to load test, pretest, or posttest.
TXWNG_UNABLE_TO_OPEN_FILE	Unable to open file.
TXWNG_FREE_NOT_MALLOCED	Attempt to use TX_Free on memory not allocated by TX_Malloc.
TXWNG_EDSEQ_NO_CHANGES_MADE	No changes made in Sequence Editor.
TXWNG_USER_CANCELED	User issued cancel on pop-up dialog box.

Engine Attribute Constants

The constants listed in Table A-3 work with TX_SetEngineAttribute and TX_GetEngineAttribute to specify what information to set or retrieve. Several of these constants refer to user callback functions and these user callback functions adhere to the following prototype:

```
int CVICALLBACK UserCallback(int callbackEvent,
                             int seqId,int runTestType,
                             int resultId);
```

Table A-3. Engine Attributes Constants

Constant	Type	Meaning
TXATTR_BEFORE_RUNTEST_FUNC_PTR	void *	The callback function to run before running a pretest, test, or posttest.
TXATTR_AFTER_RUNTEST_FUNC_PTR	void *	The callback function to run after running a pretest, test, or posttest.
TXATTR_BEFORE_RUNSEQ_FUNC_PTR	void *	The callback function to run before running a sequence or subsequence.

Table A-3. Engine Attributes Constants (Continued)

Constant	Type	Meaning
TXATTR_AFTER_RUNSEQ_FUNC_PTR	void *	The callback function to run after running a sequence or subsequence.
TXATTR_BEFORE_UUTLOOP_FUNC_PTR	void *	The callback function to run at the beginning of each UUT loop iteration.
TXATTR_AFTER_UUTLOOP_FUNC_PTR	void *	The callback function to run at the end of each UUT loop iteration.
TXATTR_OPENSEQ_FUNC_PTR	void *	The callback function to run when a sequence opens.
TXATTR_SAVESEQ_FUNC_PTR	void *	The callback function to run when a sequence saves.
TXATTR_CLOSESEQ_FUNC_PTR	void *	The callback function to run when a sequence closes.
TXATTR_BREAKPOINT_FUNC_PTR	void *	The callback function to run each time the Test Executive encounters a breakpoint.
TXATTR_ABORT_TEST_FLAG	int	Abort test flag (**Abort** button on the front panel).
TXATTR_ABORT_LOOP_FLAG	int	Abort loop flag (**Abort Loop** button on the front panel).
TXATTR_STOP_LOOP_ON_FAIL	int	Indicates whether to stop looping when a test fails.
TXATTR_STOP_SEQ_ON_TEST_FAIL	int	Indicates whether to stop sequence execution when a test fails.

Table A-3. Engine Attributes Constants (Continued)

Constant	Type	Meaning
TXATTR_BREAKPOINT_STATE	int	Current breakpoint state. Valid values: TXBP_NO_STEP TXBP_STEPINTO TXBP_STEPOVER TXBP_FINISH_SEQ
TXATTR_BREAKPOINT_AT_FIRST_TEST	int	Breakpoint before the first test.
TXATTR_FIX_SEQUENCE_PATHS	int	Indicates whether to fix the pathnames within a sequence when the sequence moved. Valid values: TRUE, FALSE, PROMPT
TXATTR_SAVE_AFTER_FIXPATHS	int	Indicates whether to save the changes after fixing the paths Valid values: TRUE, FALSE, PROMPT
TXATTR_DB_CREATE_TABLES	int	Indicates whether to create the database tables if they do not exist. Valid values: TRUE, FALSE, PROMPT.
TXATTR_OPERATOR	char *	Current operator name (set by GUI).
TXATTR_UUTNUM	char *	Current UUT number (set by GUI).
TXATTR_DEFAULT_DIR	char *	Default sequence directory.
TXATTR_DB_ENABLE_SAVE_RESULTS	int	Indicates whether to save results in a database.
TXATTR_ERR_REPORT_OUTPUT	int	Error reporting style. Valid values: 1: Report in pop-up dialog box 2: Beep on error 4: Print in standard IO window 8: CVI breakpoint You can OR these values together to achieve multiple behaviors.

Table A-3. Engine Attributes Constants (Continued)

Constant	Type	Meaning
TXATTR_TEST_ERROR_CAUSE_ABORT	int	Indicates whether a run-time error in a test causes an abort of the sequence.
TXATTR_HOOK_PTR	void *	Hook for user modifications.
TXATTR_HOOK_SIZE	int	Number of bytes the hook pointer points to.
TXATTR_AUTO_LINK_TO_SOURCE	int	Forces LoadExternalModule to use the source of a test file if the source is included in the project, default TRUE(1).
TXATTR_ENGINE_UIR_DIR	char *	Sets the directory where the Test Executive Engine user interface files are located.

Sequence Attribute Constants

The constants listed in Table A-4 work with `TX_SetSeqAttribute` and `TX_GetSeqAttribute` to specify what information to set or retrieve. Similar constants, which begin with `INFO` rather than `TXATTR`, exist for the functions `TX_SetEngineInfo` and `TX_GetEngineInfo` which are only relevant to legacy tests from previous versions of the LabWindows/CVI Test Executive.

Table A-4. Sequence Attribute Constants

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
`TXATTR_SEQPRETEST_NAMES`	char * [4]	The names associated with the sequence pretest: file name, function name, test name, and test description. **Note:** Use `TX_Free` to free returned memory.
`TXATTR_SEQPOSTTEST_NAMES`	char * [4]	The names associated with the sequence posttest: file name, function name, test name, and test description. **Note:** Use `TX_Free` to free returned memory.
`TXATTR_SIZE_OF_SEQUENCE`	int	The number of tests in the sequence.
`TXATTR_TEST_NUMBER`	int	Index of the current test.
`TXATTR_SEQPRETEST_PARAMETERS`	char * [1]	The parameter string for sequence pretest. **Note:** Use `TX_Free` to free returned memory.
`TXATTR_SEQPOSTTEST_PARAMETERS`	char * [1]	The parameter string for sequence posttest. **Note:** Use `TX_Free` to free returned memory.

Table A-4. Sequence Attribute Constants (Continued)

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_GENERAL_STRING	char * [1]	The sequence general purpose expansion string. **Note:** Use TX_Free to free returned memory.
TXATTR_METAPRETEST_NAMES	char * [4]	The names associated with the sequence load pretest: file name, function name, test name, and test description. **Note:** Use TX_Free to free returned memory.
TXATTR_METAPOSTTEST_NAMES	char * [4]	The names associated with the sequence unload posttest: file name, function name, test name, and test description. **Note:** Use TX_Free to free returned memory.
TXATTR_METAPRETEST_PARAMETERS	char * [1]	The parameter string for the sequence load pretest. **Note:** Use TX_Free to free returned memory.
TXATTR_METAPOSTTEST_PARAMETERS	char * [1]	The parameter string for the sequence unload posttest. **Note:** Use TX_Free to free returned memory.
TXATTR_SEQ_DESCRIPTION	char * [1]	The sequence description string. **Note:** Use TX_Free to free returned memory.
TXATTR_REPORT_FILE_NAME	char * [1]	The default name of the report file. **Note:** Use TX_Free to free returned memory.

Table A-4. Sequence Attribute Constants (Continued)

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_RPT_FILE_LOCK	short	Indicates whether the operator can change the name of the report file.
TXATTR_RPT_FILE_MODE	short	The report file mode: 0 for overwrite, 1 for append.
TXATTR_SEQ_FILE_NAME	char * [1]	The path and file name of the sequence. Used to detect when the sequence has moved. **Note:** Use TX_Free to free returned memory.
TXATTR_UNSAVED_EDITS	int	Indicates whether there are unsaved edits in the sequence.
TXATTR_SEQ_IS_BINARY	int	Indicates whether the sequence file is in the old binary format.
TXATTR_USAGE_COUNT	int	The number of times the sequence has been loaded.
TXATTR_DB_INFO	tDBOptionsPtr	A copy of the database information for the sequence. See the typedef tDBOptionsPtr in txengine.h. **Note:** Use TX_Free to free returned memory.
TXATTR_IS_TOP_LEVEL	int	Indicates whether the sequence was loaded directly by the user rather than as a subsequence.
TXATTR_PARENT_SEQ_ID	int	The sequence ID of the parent sequence or -1 if not a subsequence.

Table A-4. Sequence Attribute Constants (Continued)

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_SEQ_LOAD_MODE	short int	How to load the tests in the sequence. Valid values: STATIC_LOAD DYNAMIC_LOAD USE_TEST_LOAD_SPEC
TXATTR_IS_RUNNING	int	Indicates whether the sequence (or one of its subsequences) is currently running.
Test Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_TEST_NAMES	char * [8]	The names associated with an individual test: file name, function name, pretest file name, pretest function name. posttest file name, posttest function name, test name, and test description. **Note:** Use TX_Free to free returned memory.
TXATTR_TEST_LIMIT_HIGH	double	The upper value to compare the test measurement against.
TXATTR_TEST_LIMIT_LOW	double	The lower value to compare the test measurement against.
TXATTR_TEST_SPEC	short int	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

Table A-4. Sequence Attribute Constants (Continued)

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_TEST_RUN_MODE	short int	Run Mode for the test: RUN_MODE_NORMAL, RUN_MODE_FORCE_PASS, RUN_MODE_FORCE_FAIL, or RUN_MODE_SKIP
TXATTR_TEST_PASS_ACTION	short int	The action to perform if the current test passes: LOOP_ACTION_CONT, LOOP_ACTION_LOOP, or LOOP_ACTION_STOP
TXATTR_TEST_PARAMETERS	char * [1]	Parameter string for the current test. **Note:** Use TX_Free to free returned memory.
TXATTR_IS_GOTO	short int	A flag indicating whether the test is a goto.
TXATTR_GOTO_TARGET	int	The index in the sequence of the test which is the destination of the goto.
TXATTR_LOOP_COUNT	int	The number of times the current test has looped.
TXATTR_MAX_LOOPS	int	Maximum number of times a test can loop.
TXATTR_TEST_FAIL_ACTION	short int	The action to perform if the current test fails: LOOP_ACTION_CONT, LOOP_ACTION_LOOP, or LOOP_ACTION_STOP
TXATTR_TEST_DESCRIPTION	char * [1]	Description string for an individual test. **Note:** Use TX_Free to free returned memory.

Table A-4. Sequence Attribute Constants (Continued)

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_TEST_FAIL_MAKES_SEQ_FAIL	int	Indicates whether failure of an individual test causes the sequence to fail.
TXATTR_TEST_TYPE	short int	The type of test. Valid values: IS_TEST IS_SUBSEQ IS_GOTO
TXATTR_SUBSEQ_REPORT	short int	Subsequence report mode. Valid values: PARENT_MODE OWN_MODE SUPPRESS_MODE
TXATTR_SUBSEQ_DATABASE	short int	Subsequence database mode. Valid values: PARENT_MODE OWN_MODE SUPPRESS_MODE
TXATTR_SUBSEQ_ID	int	Subsequence ID for the specified test.
TXATTR_TEST_LOAD_MODE		Test load mode (valid only if sequence load mode is USE_TEST_LOAD_SPECS). Valid values: USE_SEQ_LOAD_SPEC STATIC_LOAD DYNAMIC_LOAD
TXATTR_SUPPRESS_REPORT	short int	Indicates whether to suppress reporting for an individual test.
TXATTR_SUPPRESS_DATABASE	short int	Indicates whether to suppress database operations for an individual test.

Table A-4. Sequence Attribute Constants (Continued)

Sequence Specific Attributes		
Constant	**Type**	**Meaning**
TXATTR_HAS_BREAKPOINT	short int	Indicates whether there is a breakpoint on the test.
TXATTR_PRETEST_PARAMETERS	char * [1]	The parameter string for the current test's pretest. **Note:** Use TX_Free to free returned memory.
TXATTR_POSTTEST_PARAMETERS	char * [1]	The parameter string for the current test's posttest. **Note:** Use TX_Free to free returned memory.

Result Attribute Constants

The constants listed in Table A-5 work with TX_SetResultAttribute and TX_GetResultAttribute to specify what information to set or retrieve.

Table A-5. Result Attribute Constants

Constant	Type	Meaning
TXATTR_RESULT_TEST_TYPE	int	One of the following test types: TXSR_TEST_TYPE_SEQ TXSR_TEST_TYPE_PRETEST TXSR_TEST_TYPE_TEST TXSR_TEST_TYPE_POSTTEST TXSR_TEST_TYPE_OPENTEST TXSR_TEST_TYPE_CLOSETEST
TXATTR_RESULT_SEQ_ID	int	Sequence ID.
TXATTR_RESULT_SEQ_SUBTEST	int	Test number within sequence.
TXATTR_RESULT_START_TIME	char *	Test start time.
TXATTR_RESULT_TEST_TIME	int	Test time.

Table A-5. Result Attribute Constants (Continued)

Constant	Type	Meaning
TXATTR_RESULT_TEST_RESULT	int	One of the following test results: FAIL, PASS, SKIP, AGAIN, ABORT.
TXATTR_RESULT_TEST_MEAS	double	Measurement returned by test.
TXATTR_RESULT_TEST_INBUF	char *	Test input buffer.
TXATTR_RESULT_TEST_OUTBUF	char *	Test output buffer.
TXATTR_RESULT_TEST_HOOK	void *	Address of hook pointer passed to test.
TXATTR_RESULT_TEST_ERR_FLAG	int	Test run-time error flag.
TXATTR_RESULT_TEST_ERR_LOC	int	One of the following run-time error locations: IN_NONE IN_META_PRETEST IN_SEQ_PRETEST IN_PRETEST IN_TEST IN_POSTTEST IN_SEQ_POSTTEST IN_META_POSTTEST
TXATTR_RESULT_TEST_ERR_CODE	int	Test run-time error code.
TXATTR_RESULT_TEST_ERR_MSG	char *	Test run-time error message.
TXATTR_RESULT_LOOP_COUNT	int	Loop number of test.
TXATTR_RESULT_STEP_NUMBER	int	Step number in overall sequence execution.

Menu List Attribute Constants

The constants listed in Table A-6 work with `TX_GetMenuListAttribute` and `TX_SetMenuListAttribute` to specify what information to set or retrieve.

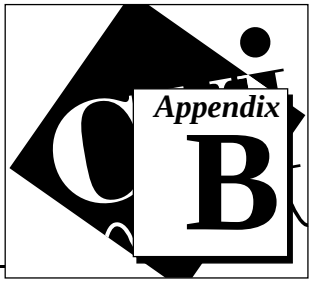
Table A-6. Menu List Attribute Constants

Menu List Attributes		
Constant	**Type**	**Meaning**
`ATTR_MENULIST_MAX_NUM_ITEMS`	int	The maximum number of items in a menu list. (When you try to add a new item to the bottom of a list that already has the maximum number of items, the new item is ignored.)
`ATTR_MENULIST_UPPER_SEPARATOR`	int	Draw a separator above menu list.
`ATTR_MENULIST_LOWER_SEPARATOR`	int	Draw a separator below menu list.
`ATTR_MENULIST_ONE_CHECK_ITEM`	int	Allow only one menu list item to be checked.
`ATTR_MENULIST_CHECK_WHEN_ADDED`	int	Automatically check item when added to menu list.
`ATTR_MENULIST_ALLOW_DUPLICATE_ITEMS`	int	Allow duplicate menu item names.
`ATTR_MENULIST_MENUBAR_HANDLE`	int	Get the menu bar handle associated with menu list (for get operations only).
`ATTR_MENULIST_MENU_ID`	int	The menu ID the menu list appears on.
`ATTR_MENULIST_BEFORE_MENU_ITEM`	int	The menu item the menu list appears above.

Table A-6. Menu List Attribute Constants (Continued)

Menu List Attributes		
Constant	**Type**	**Meaning**
ATTR_MENULIST_CALLBACK_FUNCTION	MenuList Callback Ptr	The menu list callback function.
ATTR_MENULIST_APPEND_SHORTCUT	int	Automatically add two underscores and a number, "__*X*," to create shortcut keys for menu items. *X* increments from "1" to "Z".

Menu List Item Attributes		
Constant	**Type**	**Meaning**
ATTR_MENULIST_ITEM_MENU_ID	int	The menu item ID of the menu list item (for get operations only).
ATTR_MENULIST_ITEM_NAME	char *	The menu item name of the menu list item.
ATTR_MENULIST_ITEM_NAME_LENGTH	int	The menu item name length (for get operations only).
ATTR_MENULIST_ITEM_CALLBACK_DATA	void *	The callback data for the menu list item.
ATTR_MENULIST_ITEM_CHECKED	int	Specifies whether the menu list item is checked.



Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at one of the following sites:

United States: (512) 794-5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

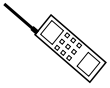
E-Mail Support (currently U.S. only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

support@natinst.com

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	 Telephone	 Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 573 4815	03 573 4816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 794 0100	512 794 8411

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. When you complete this form accurately before contacting National Instruments for technical support, our applications engineers can 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 (include version number) _____

Clock 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 reproduce the problem:_____

_____

_____

_____

_____

LabWindows/CVI Test Executive Toolkit Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. When you complete this form accurately before contacting National Instruments for technical support, our applications engineers can answer your questions more efficiently.

National Instruments Products

DAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

LabWindows/CVI version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

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: July 1997

Part Number: 320863B-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 _____

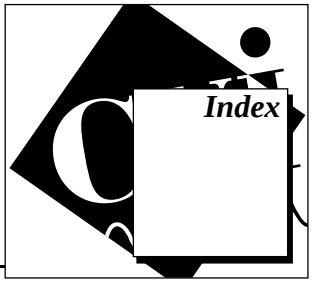
Address _____

_____

Phone (___)_____ Fax (___)_____

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

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



A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Figures

Tables