



LabVIEW™

Test Executive Reference Manual

Worldwide Technical Support and Product Information

www.natinst.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, China 0755 3904939, 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,
India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00,
Singapore 2265886, Spain (Madrid) 91 640 0085, Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@natinst.com.

© Copyright 1993, 1999 National Instruments Corporation. All rights reserved.

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 document 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 THEREOF 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

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

Product and company names mentioned herein 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 for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human. Applications of National Instruments products involving medical or clinical treatment can create a potential for death or bodily injury caused by product failure, or by errors on the part of the user or application designer. Because each end-user system is customized and differs from National Instruments testing platforms and because a user or application designer may use National Instruments products in combination with other products in a manner not evaluated or contemplated by National Instruments, the user or application designer is ultimately responsible for verifying and validating the suitability of National Instruments products whenever National Instruments products are incorporated in a system or application, including, without limitation, the appropriate design, process and safety level of such system or application.

Contents

About This Manual

Conventions	xiii
Related Documentation.....	xiv

Chapter 1 Introduction

Overview.....	1-1
Features.....	1-1
Available Packages.....	1-2
Development System	1-2
Run-Time System	1-2
Test Executive Architecture.....	1-2
System Callback VIs	1-3
Sequence Callback VIs.....	1-4
Execution Model.....	1-4
Operating Levels	1-7

Chapter 2 Getting Started

Running a Test Sequence.....	2-1
Starting the Test Executive.....	2-1
Opening and Running a Test Sequence.....	2-2
Changing to Technician Level	2-3
Executing Individual Steps and Using Single Pass Mode	2-3
Quitting the Test Executive.....	2-4
Examining a Test Program	2-4
Editing a Test Sequence.....	2-5
Example Sequences	2-9

Chapter 3 Operating the Test Executive

Controls.....	3-1
Open	3-1
Close	3-1
Quit	3-2
Login.....	3-2
Edit	3-2

New	3-2
Test UUT.....	3-2
Single Pass	3-2
Abort	3-3
Abort Loop.....	3-3
Run Step(s).....	3-3
Loop Step(s).....	3-3
Stop On Any Failure	3-4
Sequence Runtime Updates?.....	3-4
Run Mode.....	3-4
Clear Step Status	3-5
Clear Test Display.....	3-5
View Test Report	3-5
Sequence Report	3-5
Test Runtime Updates?	3-5
Operator Interface Key Assignments.....	3-6
Indicators	3-7
Sequence Display	3-7
Test Display	3-8
Result of Each Step.....	3-9
Error Messages	3-10
The Test Report	3-10
Status.....	3-12
Sequence Name.....	3-12
Sequence Information	3-12
Operator Dialog Boxes	3-13
Default Login Dialog Box.....	3-13
Default Select Sequence Dialog Box	3-13
Default UUT Information Dialog Box.....	3-14
Default Test Failed Dialog Box	3-14
Default PASS/FAIL/ABORT Banners	3-14
Run-Time Error Warning Dialog Box	3-15
Parsing Error Dialog Box.....	3-15

Chapter 4

Creating Tests and Test Sequences

Writing LabVIEW Tests.....	4-1
Required Indicators	4-2
Test Data Cluster	4-2
Error Cluster	4-3
Optional Inputs.....	4-3
Input Buffer	4-3
Invocation Information	4-4

Writing C Tests (Windows NT/98/95 and UNIX).....	4-4
Test Data Structure	4-5
Test Error Structure	4-6
Compiling Test Functions	4-7
Creating Pre-Run and Post-Run VIs	4-7
What is a Test Sequence?	4-7
What is a Step?	4-8
Creating or Editing a Test Sequence.....	4-8
Step Editing Elements	4-9
Insert.....	4-9
New Step	4-9
Copy Steps, Cut Steps, Delete Steps, Paste Steps, and Undo Step Edits.....	4-9
Using the Editing Elements.....	4-10
Adding a New Step.....	4-10
Modifying a Step	4-10
Copying a Step.....	4-10
Deleting a Step.....	4-10
Mass Editing	4-11
Step Editor Controls	4-11
Type	4-11
Name (LabVIEW Test, C Test, Sequence)	4-11
Resource (LabVIEW Test, C Test, Sequence).....	4-11
Function (C Test)	4-12
Limit Specification (LabVIEW Test, C Test).....	4-12
Load Specification (LabVIEW Test, C Test, Sequence)	4-13
Run Mode (LabVIEW Test, C Test, Sequence).....	4-14
FAIL Action (LabVIEW Test, C Test, Sequence).....	4-14
Max Loop Count.....	4-14
Input Buffer? (LabVIEW Test, C Test)	4-14
Invocation Info? (LabVIEW Test).....	4-15
Show Test VI Panel at Runtime? (LabVIEW Test).....	4-15
Edit Test VI (LabVIEW Test).....	4-15
Edit Dependencies.....	4-15
Edit Step Comment (LabVIEW Test, C Test, GOTO, Sequence)	4-16
GOTO Target (GOTO)	4-16
GOTO Conditions (GOTO)	4-16
Sequence Options	4-16
Sequence Load Specification	4-16
Sequence Path Specification	4-17
Stop on Any Failure	4-17
Description	4-17
Enable Test Report Logging	4-17

Report File Mode	4-18
Change Report File	4-18
Sequence VIs	4-18
Sequence Errors	4-18
File Menu	4-20
Edit Menu.....	4-20
Sequence Editor Control Key Assignments	4-20
Sequence Editor Menu Shortcuts	4-22
Editing Dependencies	4-22
AND and OR Expressions	4-23
Complex Dependencies	4-24
Copy, Cut, Delete, Paste, and Undo	4-25
Dependency Editing Rules	4-25
OK	4-25
Cancel	4-25
Dependency Editor Key Assignments	4-26
Relationship among Dependencies, Run Mode, and Test Flow	4-26

Chapter 5

Modifying the Test Executive

System Configuration File, testexec.ini.....	5-1
[Callback Paths] Section	5-1
Patching Callback Paths	5-2
[Operator Interface Path] Section	5-3
[Preferences] Section	5-3
Operator Interface VI	5-4
Modifying the Default VI	5-4
Front Panel	5-4
Block Diagram	5-4
Command Loop	5-5
Callback VIs	5-6
Test Executive Callback VI Calling Interface	5-6
System Callbacks	5-6
Login.....	5-7
Select Sequence	5-8
Open Sequence	5-9
Close Sequence	5-10
Save Sequence	5-10
Sequence Report	5-11
Exit.....	5-11

Sequence Callbacks	5-12
Pre-UUT Loop	5-14
Pre-UUT	5-14
Post-UUT	5-16
Post-UUT Loop	5-17
Pre-Step and Post-Step Callbacks	5-17
Test Report	5-19
Post Run-Loop Test	5-20
Test Failure	5-21
Open Test VI	5-22
Test Executive Typedef Controls	5-23
Typedefs for Callback VIs	5-23
TYPEDEF - Login Info.ctl	5-23
TYPEDEF - Sequence.ctl	5-24
TYPEDEF - Sequence Element.ctl	5-25
TYPEDEF - UUT Results.ctl	5-27
TYPEDEF - Sequence Result.ctl	5-27
TYPEDEF - Test Result.ctl	5-28
Typedefs for LabVIEW Tests	5-29
TYPEDEF - Invocation Info.ctl	5-30
TYPEDEF - Input buffer.ctl	5-30
TYPEDEF - Mode.ctl	5-30
TYPEDEF - Test Data.ctl	5-30
Common Modifications	5-31
Changing Passwords	5-31
Changing PASS/FAIL/ABORT Banners	5-32
Changing the UUT Serial Number Prompt	5-33
Changing the Test Report	5-33
Using Another Application for Report Generation	5-34
Advanced Modifications	5-34
Result Logging Alternatives	5-34
Logging Test Results on a Per-UUT Basis	5-34
Per-UUT Logger Callback.vi	5-35
Test String Callback.vi	5-35
Logging Results to a Database Using the LabVIEW SQL Tools (Windows only)	5-35
Modifications to the System Configuration File	5-36
The Alternate Callback VIs	5-36
Using LabVIEW Test Shells	5-38
Example Sequence Using LabVIEW Test Shells	5-39

Chapter 6 Deploying the Test Executive

LabVIEW Test Executive Run-Time System	6-1
Building a Run-Time System	6-1
Other Required Components for a Complete Run-Time System	6-2
Callback and Test VIs	6-2
The testexec.ini File	6-3
Test Sequences	6-4
Shared Libraries (C Test Resources)	6-4

Appendix A Common Questions

Appendix B Sequence Conversion Notes

Version 4.0 and 5.0 Conversion	B-1
Step 1—Use the 5.0 Sequence File Converter	B-1
Controls	B-1
Indicators	B-2
Step 2—Compile Your Test VIs	B-2

Appendix C Technical Support Resources

Glossary

Index

Figures

Figure 1-1. Architecture of the Test Executive	1-3
Figure 1-2. Test Sequence Callback VIs	1-5
Figure 1-3. Flow of Execution in Test UUT Mode	1-6
Figure 1-4. Flow of Execution in Single Pass Mode	1-6
Figure 3-1. Sample Test Report	3-11
Figure 5-1. Flow of Sequence Callback VIs in a UUT Test Loop	5-13
Figure 5-2. Test VI Shell Configuration and Execution	5-41

Tables

Table 1-1.	Operating Level Capabilities.....	1-7
Table 3-1.	Default Operator Interface Key Assignments	3-6
Table 3-2.	Run Mode Field Values.....	3-7
Table 3-3.	Step Status/Result Field Values	3-8
Table 3-4.	Comparison Values and Relative Limits.....	3-9
Table 3-5.	Status Indicator Values.....	3-12
Table 4-1.	Test Data Cluster Elements	4-2
Table 4-2.	Error Cluster Elements	4-3
Table 4-3.	tTestData Structure Parameters	4-5
Table 4-4.	tTestError Structure Parameters	4-6
Table 4-5.	Comparison Type Values	4-12
Table 4-6.	Run Mode Options	4-14
Table 4-7.	FAIL Action Options.....	4-14
Table 4-8.	Possible Errors and Corrective Actions in the Sequence Errors Dialog Box	4-19
Table 4-9.	Key Assignments for Sequence Editor Controls	4-20
Table 4-10.	Sequence Editor Menu Commands	4-22
Table 4-11.	Dependency Editor Key Assignments.....	4-26
Table 4-12.	Run Mode Step Result Values.....	4-27

About This Manual

This manual describes the LabVIEW Test Executive package. You can use this add-on package for automated sequencing of test programs in LabVIEW 5.1 and later.

Conventions

The following conventions appear in this manual:

< >

Angle brackets enclose the name of a key on the keyboard—for example, <PageDown>.

-

A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-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.



This icon denotes a tip, which alerts you to advisory information.



This icon denotes a note, which alerts you to important information.

bold

Bold text denotes a parameter, menu name, palette name, menu item, return value, function panel item, or dialog box button or option.

italic

Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

monospace

Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

monospace bold

Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

Related Documentation

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

- *G Programming Reference Manual*
- *LabVIEW Function and VI Reference Manual*
- *LabVIEW QuickStart Guide*
- *LabVIEW Technical Resource**
- *LabVIEW Test Executive Release and Upgrade Notes*
- LabVIEW Tutorial
- *LabVIEW User Manual*

* To order the *LabVIEW Technical Resource* Spring 1996 issue, access the LabVIEW Technical Resource web site at www.ltrpub.com, or call 214 706 0587, Fax: 214 706 0506.

Introduction

This chapter lists the main features of the Test Executive, explains its execution model, and describes its three operating levels.

Overview

A Test Executive is an application you can use to develop and execute test sequences. A test sequence consists of a series of test programs combined with flow control statements that you use to test a unit under test (UUT). The LabVIEW Test Executive can call test programs written using both G and C (Windows NT/98/95 and UNIX only) programming languages.

The Test Executive includes a powerful Test Executive engine for performing test sequencing and sequence editing operations, an operator interface virtual instrument (VI), and a set of callback VIs for handling various interface and data-logging tasks. The operator interface and callback VIs are provided with G source code, allowing users to change and/or expand the functionality of the LabVIEW Test Executive.

Features

Some of the main features of the Test Executive are as follows:

- Runs hierarchical test sequences
- Calls test programs written in either LabVIEW's G programming language or in C (Windows NT/98/95 and UNIX only)
- Sequences tests based on PASS/FAIL states and advanced dependencies
- Logs test reports to either an ASCII file or SQL-compliant database (in conjunction with the LabVIEW SQL Tools, included in the Enterprise Connectivity Toolset).
- Contains run-time interfacing, including the ability to prompt for operator and UUT serial numbers, to display PASS/FAIL banners, and to perform run-time error notification
- Generates ASCII sequence reports to files
- Allows continuous testing in Test UUT mode

Available Packages

The Test Executive is available in two versions.

Development System

The LabVIEW Test Executive Development System is designed to run under the LabVIEW development environment. It consists of the Test Executive engine, an operator interface VI, a library of callback VIs, and a library of LabVIEW type definitions for use when developing LabVIEW tests and custom callback VIs. Example test programs and test sequences also are included with the Development System.

Run-Time System

You can use the LabVIEW Application Builder to build a LabVIEW Test Executive Run-Time System. With the Run-Time System, you can distribute the Text Executive to many test stations without incurring the expense of outfitting each station with the LabVIEW Development System.

Test Executive Architecture

The Test Executive includes an engine, an operator interface VI, and a set of *callback VIs*, which are LabVIEW VIs designed for specific interface and data-logging operations. The engine handles tasks such as creating, editing, loading, saving, and executing test sequences. The engine uses the operator interface VI and the callback VIs to handle tasks such as user login, report generation, and datalogging.

Figure 1-1 shows the relationship between the Test Executive engine, the operator interface VI, and the callback VIs.

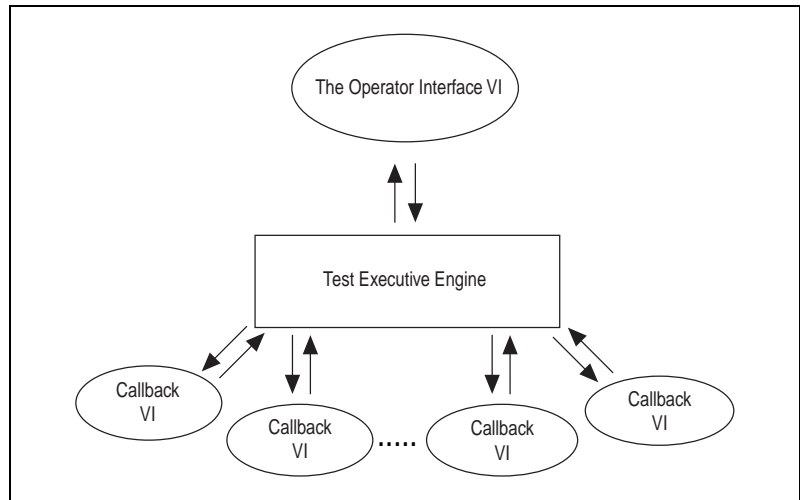


Figure 1-1. Architecture of the Test Executive

You can customize the Test Executive by modifying or replacing the operator interface VI or the callback VIs. The operator interface VI is the main panel of the Test Executive. With the main panel, you can issue commands to the Test Executive engine and see the results of those commands. The Test Executive engine works with 17 callback VIs, which are divided into *system* and *sequence* callback VIs. For information about modifying the operator interface VI, the system callback VIs, or the sequence callback VIs, refer to Chapter 5, *Modifying the Test Executive*.

System Callback VIs

The Test Executive works with the following system callback VIs:

- Close Sequence
- Exit
- Login
- Open Sequence
- Save Sequence
- Select Sequence
- Sequence Report

The Login callback VI identifies the Test Executive operator and sets the operating level. The Select Sequence callback VI prompts the operator to choose a test sequence to open. The Open, Save, and Close Sequence callback VIs are called when the operator opens, saves, or closes a test

sequence. The Sequence Report callback VI generates an ASCII report file describing the currently loaded test sequence. The Exit callback VI is called when the user exits the Test Executive.

Sequence Callback VIs

The Test Executive works with the following sequence callback VIs:

- Open Test VI
- Pre-Step
- Pre-UUT
- Pre-UUT Loop
- Post Run-Loop Test
- Post-Step
- Post-UUT
- Post-UUT Loop
- Test Failure
- Test Report

The next section describes how to use the sequence callback VIs.

Execution Model

The Test Executive can execute a sequence in one of four modes—Test UUT, Single Pass, Run Step(s), or Loop Step(s).

- Test UUT, invoked when the user clicks the **Test UUT** button, executes a test sequence repetitively. This mode is the production operating mode for testing multiple UUTs.
- In Single Pass mode, the test sequence executes only once. Single Pass mode is primarily for use during development and also can be useful for diagnostic purposes.
- Run Step(s) mode, invoked when the user clicks the **Run Step(s)** button, executes the steps currently selected in the Sequence Display.
- Loop Step(s) mode, invoked when the user clicks the **Loop Step(s)** button, executes the step currently selected in the Sequence Display a specified number of times (or, if only a single step is selected, until that step fails).

Both Run Step(s) and Loop Step(s) modes are intended primarily for diagnostic purposes.

Every test sequence has a set of sequence callback VIs that handle certain run-time or edit-time events. When you create a new test sequence, the Test Executive gives it a default set of sequence callback VIs. You can override these default callback VIs to change the way the Test Executive executes or edits a test sequence. The Sequence Editor also calls these VIs when you want to edit a test VI. Figure 1-2 shows a test sequence and its associated callback VIs.

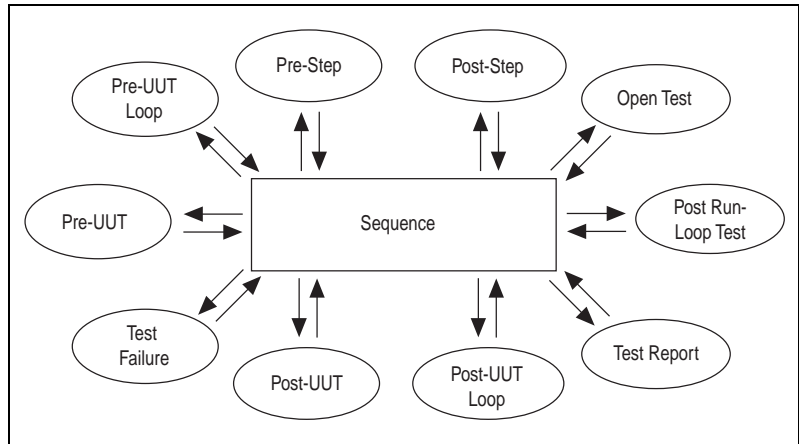


Figure 1-2. Test Sequence Callback VIs

For a typical test sequence in Test UUT mode, the Test Executive engine calls the Pre-UUT Loop callback VI before testing the first UUT. The Pre-UUT Loop callback VI performs user-specified setup or data logging operations. Then, before testing each new UUT, the Test Executive calls the Pre-UUT callback VI, which by default prompts the operator for UUT information.

After testing a UUT, the Test Executive calls the Post-UUT callback VI, which by default displays a PASS/FAIL banner. After testing the last UUT, the Test Executive calls the Post-UUT Loop callback VI for user-specified cleanup or data-logging operations. Then, the Test Executive engine calls the Test Report callback VI, which by default generates a spreadsheet-style

ASCII report detailing the test results for each UUT tested. Figure 1-3 shows the overall flow of execution in Test UUT mode.

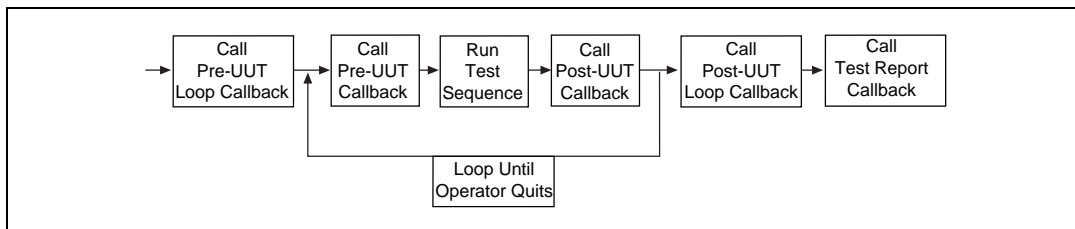


Figure 1-3. Flow of Execution in Test UUT Mode

In Single Pass mode, the Test Executive engine does not call the Pre-UUT Loop, Pre-UUT, Post-UUT, or Post-UUT Loop callback VIs. Figure 1-4 shows the overall flow of execution in Single Pass mode.

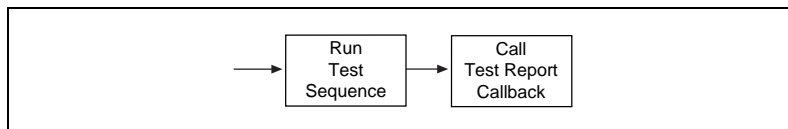


Figure 1-4. Flow of Execution in Single Pass Mode

When a step with a FAIL Action of `Callback` fails, the Test Executive engine calls the Test Failure callback VI to determine what action to take.

After you click the **Run Step(s)** or **Loop Step(s)** button to run or loop the individual steps selected in the Sequence Display, the Test Executive engine calls the Post Run-Loop Test callback VI.

In all execution modes, the Pre-Step and Post-Step callback VIs are called before and after each test step, respectively.

In contrast to the other sequence callback VIs, which are called by the Test Executive engine, the Sequence Editor calls the Open Test callback VI when you click the **Edit Test VI** button. The default Open Test callback VI opens the front panel of the current LabVIEW test.

In addition to the sequence callback VIs, each sequence may specify a Pre-Run and/or Post-Run VI. These VIs run before and after their associated sequence runs in either Test UUT or Single Pass mode. They also run before and after any group of steps you run from their associated test sequence in either Run Step(s) or Loop Step(s) mode. You specify Pre-Run and Post-Run VIs in the Sequence Options dialog box of the Sequence Editor.

Operating Levels

The Test Executive has three operating levels—Developer, Technician, and Operator. Table 1-1 summarizes the capabilities available in each operating level.

Table 1-1. Operating Level Capabilities

Level	UUT Test	Single Pass Run Step(s) Loop Step(s)	Edit Sequences
Developer	Yes	Yes	Yes
Technician	Yes	Yes	No
Operator	Yes	No	No

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

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

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

The Login callback VI determines the initial operating level. When you first load the Test Executive, the default Login callback VI displays a dialog box that prompts you for a password, which sets the operating level. To change the operating level when running the Test Executive, you can re-open the Login dialog box by clicking the **Login** button on the main panel of the Test Executive.

Getting Started

This chapter introduces the basic concepts of Test Executive operation and test sequence development and contains the following examples:

- Running a Test Sequence
- Examining a Test Program
- Editing a Test Sequence

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

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



Note This chapter assumes that you are running the Test Executive Development System. If you are executing the Test Executive Run-Time System, you cannot run the *Examining a Test Program* example.

Running a Test Sequence

This section describes how to run a test sequence.

Starting the Test Executive

Perform the following steps to start the Test Executive.

1. Launch LabVIEW.
2. Select **Project>Test Executive>Launch Test Executive...** to run the default operator interface VI.

When the Test Executive begins running, it loads the callback VIs into memory. After loading the system callback VIs, it calls the Login callback VI, which displays the Login dialog box.

3. Type your name and password into the Login dialog box. Use the <Tab> key to move from the **Name** to the **Password** field.

The default Login callback VI uses the password you enter to set the operating level. It then passes this operating level to the Test Executive, which configures itself accordingly. In this session, you run the Test Executive at the Operator level. You access the Operator level by typing any text in the **Password** field.



Note To access the other operating levels, you must type `technician` for the Technician level and `developer` for the Developer level.

4. Press <Enter> (<return>, <Return>) or click the **OK** button to confirm your entries.

On the Test Executive front panel, notice the word **STOPPED** that appears directly above the Sequence Display. This field is the status indicator. A flag of **STOPPED** indicates that no test sequence is currently running.

Opening and Running a Test Sequence

Perform the following steps to open and run a test sequence.

1. Click the **Open** button in the upper left corner of the operator interface panel.
2. Select the file `COMPUTER.SEQ`, which is located in the `EXAMPLES` directory of the Test Executive installation.

A short delay occurs while the Test Executive loads the VIs that the sequence requires into memory. After you load `COMPUTER.SEQ`, notice that the test sequence appears in the Sequence Display. The name of the test sequence and the sequence description also appear at the top of the Test Executive front panel.

3. Click the **Test UUT** button located beneath the Sequence Display to execute the test sequence. The Test Executive calls the default Pre-UUT callback VI, which prompts you to enter the serial number of the UUT.
4. Type any value for the serial number and press <Enter> (<return>, <Return>) or click the **OK** button.

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

- The Status indicator displays **RUNNING**.
- As each step runs, the word **RUNNING** appears next to the name of the active step in the Sequence Display.

- After each step runs, the PASS/FAIL status of the step appears next to the name of the step in the Sequence Display, and the step result appears in the Test Display.

When the sequence execution completes, the Test Executive calls the default Post-UUT callback VI, which displays a PASS/FAIL banner.

5. Click the **OK** button in the PASS/FAIL banner. The Test Executive begins the next test cycle by calling the default Pre-UUT callback VI again. The Test Executive continues to cycle to the next UUT until you click the **Stop** button in the UUT Information dialog box. At that point, the Test Executive exits the Test UUT loop and calls the Test Report callback VI to generate the test report.
6. To view the test report after execution completes, click the **View Test Report** button. The report appears in the Test Display string indicator.

The Test Report includes the name and description of the sequence, the date and time that testing ended, the name of the user, and the results of testing for each UUT.

Changing to Technician Level

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

1. Click the **Login** button on the Test Executive front panel.
2. In the Login dialog box, type the word `technician` in the **Password** field and click the **OK** button. The **Single Pass**, **Run Step(s)**, and **Loop Step(s)** buttons appear in the lower left corner of the Test Executive front panel.
3. If you do not see these buttons, click the **Login** button again and retype the word `technician` in the **Password** field. Remember that the password is case sensitive.

Executing Individual Steps and Using Single Pass Mode

To run an individual step, click the name of the desired step in the Sequence Display and click the **Run Step(s)** button. Notice that only the selected step runs. You can select multiple steps to run in Run Step(s) mode by <Shift>-clicking them in the Sequence Display. Run Step(s) mode selectively runs individual steps for diagnosis and troubleshooting. The status of each step appears next to the step names in the Sequence Display. To execute steps repeatedly, click the **Loop Step(s)** button.

Now, click the **Single Pass** button. Clicking this button runs the entire test sequence once. When running in Single Pass mode, the Test Executive skips all sequence callback VIs except for the Test Report callback VI. Notice that the sequence executes one time and then stops. When you click the **View Test Report** button, the Test Executive displays the Test Report as it did at the Operator level.

Quitting the Test Executive

Click the **Quit** button to stop execution of the Test Executive. When the Test Executive stops running, LabVIEW automatically quits if you are logged in at the Operator or Technician levels. Launch the LabVIEW full development system before proceeding to the next examples.

Examining a Test Program

In this section, you examine a sample test program written in LabVIEW. You need this information only if you plan to write LabVIEW tests and incorporate them into test sequences. You must be familiar with the LabVIEW full development environment to complete this example. For detailed information about the topics in this section, see Chapter 4, *Creating Tests and Test Sequences*.

Perform the following steps to learn how to build a LabVIEW test.

1. Launch LabVIEW, if you have not already done so.
2. Open `random.vi`, located in `TESTS\TEST_VIS.LLB` in the Test Executive installation directory.

`random.vi` illustrates the basic structure of a LabVIEW test that the Test Executive runs. The front panel of the VI contains two clusters—the Test Data cluster and the error out cluster.

The Test Data cluster transmits information about the result of the test. The error out cluster transmits information the Test Executive uses for run-time error handling. In this example, use `random.vi` as if it were a new test VI to step through the test sequence creation process.

3. Open the block diagram of `random.vi` by selecting **Windows»Show Diagram** from the menu on the front panel.

This VI generates two random numbers, **Limit** and **Measurement**. The VI compares **Limit** to **Measurement**, setting the PASS/FAIL flag in the Test Data cluster to the result of the comparison. This VI also passes one of the random numbers and a comment as the Numeric Measurement and

Comment elements of the Test Data cluster. The Test Executive uses these elements when you create a test sequence that calls this VI.

4. Close `random.vi`. Do not save any changes.

Editing a Test Sequence

This section describes how to set up and edit a test sequence.

1. Start the Test Executive.
 - a. Launch LabVIEW if you have not already done so.
 - b. Select **Project>Test Executive>Launch Test Executive...** to run the default operator interface VI. Type the word `developer` in the **Password** field of the Login dialog box and click the **OK** button. A row of six buttons appears in the upper left corner of the operator interface front panel. If you do not see six buttons, click the **Login** button and retype `developer` as the password. Remember that the password is case sensitive.
2. Edit the Sequence
 - Click the **Edit** button to invoke the Sequence Editor.

The first time you invoke the Sequence Editor, there is a delay while the Test Executive loads the editor. This loading delay occurs only the first time you open the Sequence Editor. Notice that the list box at the top of the Sequence Editor panel is empty. This list box, called the *sequence list*, shows the defined steps for the current sequence. With the Sequence Editor, you input all of the specifications required to define a test sequence.
3. Create a Step.

Complete the following steps to add `random.vi` to the sequence.

 - a. Click the **New Step** button to add a new, untitled step to the top of the sequence list. Notice that the edit controls are displayed and that the edit control `Name` is automatically selected. Also notice that the ring control `Type` is set to `LabVIEW Test`. Because this example involves only LabVIEW-based test programs, do not change the `Type` setting.
 - b. Type the name `Random-Boolean` into the `Name` control and press `<Enter>` (`<return>`, `<Return>`). Notice that `Random-Boolean` now appears in the sequence list.
 - c. Click the **Select Resource...** button to choose the VI that you want to run for this step. For this example, select `random.vi`. Notice

that the Resource control is automatically filled with the VI path you selected.

4. Configure the Limit Specification.

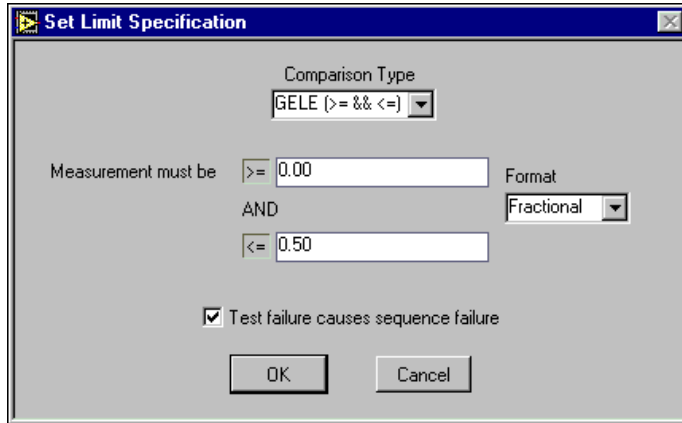
To determine if a step passes or fails, the Test Executive must have a limit specification. The Test Executive looks at the Test Data cluster of the test VI and applies the limit checking you specify in the Sequence Editor to that data. The Test Data cluster contains a Boolean flag, a numeric measurement, and a string measurement. You can use any of these elements to determine if the VI passes.

- a. Click the **Set Limit Specification** button to view the Set Limit Specification dialog box. Click the **Comparison Type** ring control to see the available types of checking. If you choose a numeric comparison, you must enter the numeric limits used for the comparison. If you choose a string comparison, you must enter a reference string to be used for the comparison. For this example, set Comparison Type to `Boolean`.
- b. Click the **OK** button to confirm the limit specification. Notice that the Limit Specification control now contains the text `{BOOL}`.

5. Add Another Step

Now, perform the following steps to add another step below Random-Boolean and change the limit specification to numeric.

- a. Make sure that the Insert control is set to `below` and click the **New Step** button. The new step appears below the currently selected step in the sequence list. Name the new step `Random-Numeric`.
- b. Click the **Select Resource...** button and again select `random.vi`. Then, click the **Set Limit Specification** button and set the Comparison Type to numeric comparison, `GELE (>= && <=)`, which means the numeric value returned by the test VI must be greater than or equal to a lower limit *and* less than or equal to an upper limit. Set the lower limit to `0.00` and the upper limit to `0.50`, as shown in the following illustration.



With the Format control, you set the numeric limits to fractional, scientific, decimal, hexadecimal, octal, or binary notation. For this example, use fractional notation.

- c. Click the **OK** button to accept the limit specifications. Your completed test sequence appears in the sequence list of the Sequence Editor dialog box.
 - d. To modify the definition of a step, click the step you want to modify in the sequence list. The specifications of the step appear in the edit fields. Make any changes to these edit fields, and the Test Executive automatically applies the changes to the step.
6. Set Dependencies.

Next, set up a dependency between the two steps in your sequence.

- a. Click the **Edit Dependencies** button, which opens the Dependency Editor.
- b. To set up a dependency between the two steps, you must specify that `Random-Boolean` must pass for `Random-Numeric` to execute. If `Random-Numeric` is not selected in the step list box, select it. Notice that the **Dependencies** list box is empty, indicating that `Random-Numeric` has no dependencies.

- c. To add the desired dependency, select `Random-Boolean` in the **New Determinants** list box and click the » button. This adds a `FAIL` dependency on `Random-Boolean` to the Dependency list for `Random-Numeric`. (Double-clicking `Random-Boolean` also allows you to add this `FAIL` dependency.) Change the `FAIL` dependency to a `PASS` dependency by clicking the **Change to PASS** button. (Double-clicking the `FAIL` dependency also allows you to change this dependency from `FAIL` to `PASS`.)
 - d. Click the **OK** button to keep the new dependencies and return to the Sequence Editor.
7. Run the Sequence
- You are now ready to run your test sequence.
- a. Return to the main Test Executive front panel by selecting **File»Exit**. After you save the new sequence, it appears in the **Sequence Display** list box.
 - b. Click the **Test UUT** button to run the sequence. The Test Executive automatically determines the `PASS/FAIL` status based on the values placed in the Test Data cluster. Perform the following steps to view your specification.
 - After you test several UUTs, click the **Stop** button in the UUT Information dialog box.
 - To see the data generated by `random.vi` for each test, click the **View Test Report** button.
8. Quit the Test Executive from the Developer Level
- Click the **Quit** button to quit the Test Executive. The Test Executive prompts you to confirm or cancel the `Quit` operation. Proceed with the operation, and the Test Executive automatically prompts you to save the sequence you created.

When you run the Test Executive at the Developer level, the application stays in memory after it finishes executing.

If you followed the examples presented in this chapter, you now know how to operate the Test Executive, develop test programs using LabVIEW, and use the Sequence Editor to create sequences that use these VIs. These are the fundamental steps required to create test sequences that run in the Test Executive. The remaining chapters of this manual describe the capabilities of the Test Executive in greater detail.

Example Sequences

The Test Executive package includes nine test sequence examples located in the `EXAMPLES` subdirectory of the Test Executive installation directory. If you quit the Test Executive from the Operator or Technician level, you need to restart the Test Executive to see these examples. The sequences, listed below, demonstrate different aspects of the Test Executive.

- `COMMENT.SEQ` executes tests that use the Comment field to log test results in a customized format. When you run `COMMENT.SEQ`, notice that test results in the Test Report contain multiple, custom-formatted lines rather than the standard formatted lines.
- `COMPUTER.SEQ` contains a sequence that includes Pre-run and Post-run VIs (VIs that run before or after a test sequence), multiple dependencies, and tests that use a variety of comparison types.
- `RTERROR.SEQ` contains the same tests as `COMPUTER.SEQ` but generates a run-time error during the test to illustrate the Run-time Error dialog boxes.
- `UNDEFINE.SEQ` demonstrates how the Test Executive handles sequence files containing invalid information. This condition is called a parsing error. The Parsing Error dialog box opens when you try to load `UNDEFINE.SEQ`. `UNDEFINE.SEQ` is not meant to run. Its purpose is to show how the Test Executive handles a parsing error.
- `comp_new.seq` contains a sequence that calls subsequences.
- `cpu_lv.seq` contains a sequence that is called by `comp_new.seq`.
- `cpu_diag.seq` contains a sequence that is called by `comp_new.seq`.
- `computer_cvi.seq` (Windows NT/98/95 only) contains a sequence that calls tests developed in LabWindows/CVI.
- `cpu_cvi.seq` (Windows NT/98/95 only) contains a sequence that is called by `computer_cvi.seq`.

Operating the Test Executive

This chapter describes the operation of the main Test Executive front panel—the controls, indicators, and operator dialog boxes. The main front panel is the user interface for both development and run-time operation. When at the Technician and Operator levels, the Test Executive disables some of the buttons.

Controls

The controls on the Test Executive front panel access the following three areas of operation:

- Sequence file operations and login
- Execution
- Display

The rest of this section describes the purpose of each control, grouping them according to their areas of operation.

Open



<F2>

The **Open** button invokes the Select Sequence callback VI for selecting a test sequence to load into memory from a file. Selecting a valid sequence file opens the sequence and loads the step resource and sequence callback VIs into memory. A delay occurs proportional to the number and size of the step resources being loaded. The **Open** button is visible at all operating levels. The default key assignment for **Open** is <F2>.

Close



<F3>

The **Close** button closes the current sequence visible in the Sequence Display and unloads the step resources and the sequence callback VIs from memory. A delay occurs proportional to the number and size of the step resources being unloaded. The **Close** button is visible at all operating levels. The default key assignment for **Close** is <F3>.

Quit



<F10>

The **Quit** button causes the Test Executive to stop execution. After you click the **Quit** button, the Test Executive prompts you to confirm or cancel the Quit operation. The default key assignment for **Quit** is <F10>.



Note If the Test Executive is at the Operator or Technician operating level, clicking the **Quit** button stops the Test Executive and quits LabVIEW.

Login



<F4>

The **Login** button calls the Login callback VI. The default Login callback VI displays a Login dialog box for entering your name and/or password. The **Login** button is visible at all operating levels. The default key assignment for **Login** is <F4>.

Edit



<F5>

The **Edit** button invokes the Sequence Editor. There is a short loading delay when you first open the Sequence Editor. The **Edit** button is visible only when the Test Executive is running at the Developer operating level. The default key assignment for **Edit** is <F5>.

New



<F6>

The **New** button loads a new Untitled sequence, displaying it in the Sequence Display. You can have only one Untitled sequence loaded at a time. The **New** button is visible only when the Test Executive is at the Developer operating level. The default key assignment for **New** is <F6>.

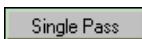
Test UUT



<Shift-F1>

The **Test UUT** button initiates repetitive execution of the currently visible test sequence for UUT testing. (See the [Execution Model](#) section of Chapter 1, [Introduction](#), for information about the Test UUT mode of execution.) The **Test UUT** button is visible at all operating levels. The default key assignment for **Test UUT** is <Shift-F1>.

Single Pass



<Shift-F2>

The **Single Pass** button initiates a single execution of the currently visible test sequence. (See the [Execution Model](#) section of Chapter 1, [Introduction](#), for information about the Single Pass mode of execution.) The **Single Pass** button is visible only at the Developer and Technician operating levels. The default key assignment for **Single Pass** is <Shift-F2>.

Abort



The **Abort** button stops sequence execution after the currently executing step completes. If you click the **Abort** button while in Test UUT mode, the Test Executive stops testing on the current UUT and proceeds to the next UUT. The default Post-UUT callback VI displays an ABORT banner when you abort testing. The **Abort** button is visible at all operating levels but is active only when a test is running. The default key assignment for **Abort** is <Shift-F10>.



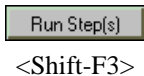
Note The Test Executive engine does not log complete result information for aborted sequences. Therefore, the test report that the Text Executive generates will contain incomplete result information if you abort the execution of a test sequence.

Abort Loop



The **Abort Loop** button appears only when the Test Executive loops on a failing step. Clicking the **Abort Loop** button stops the loop execution. Test sequence execution then continues with the next step. The **Abort Loop** button appears at all operating levels. The default key assignment for **Abort Loop** is <Shift-F9>.

Run Step(s)



The **Run Step(s)** button executes the steps currently selected in the Sequence Display. After the step runs, the Test Executive calls the Post Run-Loop Test Callback VI. The **Run Step(s)** button is visible only at the Developer and Technician operating levels. The default key assignment for **Run Step(s)** is <Shift-F3>.

Loop Step(s)



The **Loop Step(s)** button initiates repetitive execution of the step currently selected in the Sequence Display. The **Loop Step(s)** button is visible only at the Developer and Technician operating levels. When you select **Loop Step(s)**, the Loop Parameters dialog box appears.

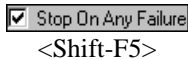
With the Loop Parameters dialog box, you can either execute a specific number of iterations or loop until a step fails (if you have selected only one step for execution). To specify a number of iterations, make sure the **Loop until FAIL** checkbox is not selected and enter the number of iterations in the **Loop** field. To loop until a step fails, select the **Loop until FAIL** checkbox. To confirm your inputs, click the **OK** button. To cancel, click the **Cancel** button. After the steps finish looping, the Test Executive calls the

Post-Run Loop Test callback VI. The default key assignment for **Loop Step(s)** is <Shift-F4>.



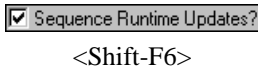
Note When you run steps using the **Run Step(s)** or **Loop Step(s)** button, the Test Executive disregards the Run Mode setting for the selected step and forces the step to run normally. In contrast, if the Run Mode setting of a test is Skip, Force PASS, or Force FAIL in Test UUT or Single Pass mode, the Test Executive does not run the step. Instead, it generates a step result of Skip, PASS, or FAIL.

Stop On Any Failure



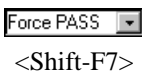
If you select the **Stop On Any Failure** checkbox, the Test Executive stops executing the current sequence whenever a step in that sequence fails. When you clear this checkbox, the Test Executive runs the current sequence as normal. Changes to this box do not affect the default sequence setting for **Stop On Any Failure**. You must change this default setting in the Sequence Editor. The **Stop On Any Failure** checkbox is visible at all operating levels. The default key assignment for **Stop On Any Failure** is <Shift-F5>.

Sequence Runtime Updates?



When you select the **Sequence Runtime Updates?** checkbox, the Test Executive updates the Sequence Display whenever a new step starts or finishes running, or another test sequence runs. When you unselect this box, the Test Executive does not update the Sequence Display. To make the Test Executive run as fast as possible, disable Sequence Display updates by clearing this checkbox. The **Sequence Runtime Updates?** checkbox is visible at all operating levels. The default key assignment for **Sequence Runtime Updates?** is <Shift-F6>.

Run Mode



The **Run Mode** ring control displays the run mode setting for the currently selected step. You also use this control to change the run mode of a step for diagnostic purposes. Changes made to the run mode of any step using this control do not affect the default run mode of the step. You must change the default value in the Sequence Editor. The **Run Mode** control is visible only at the Developer and Technician operating levels. The default key assignment for **Run Mode** is <Shift-F7>.


Clear Step Status



<Ctrl-F1>

The **Clear Step Status** button clears the Step Status/Result field for each step in the Sequence Display. The **Clear Step Status** button is visible at all operating levels. The default key assignment for **Clear Step Status** is <Ctrl-F1> (<command-F1>, <meta-F1>, <Alt-F1>).

Clear Test Display



<Ctrl-F2>

The **Clear Test Display** button clears the contents of the Test Display. The **Clear Test Display** button is visible at all operating levels. The default key assignment for **Clear Test Display** is <Ctrl-F2> (<command-F2>, <meta-F2>, <Alt-F2>).

View Test Report



<Ctrl-F3>

The **View Test Report** button displays the current Test Report in the Test Display. The **View Test Report** button is visible at all operating levels. The default key assignment for **View Test Report** is <Ctrl-F3> (<command-F3>, <meta-F3>, <Alt-F3>).

Sequence Report



<Ctrl-F4>

The **Sequence Report...** button invokes the Sequence Report callback VI. The default Sequence Report callback VI generates a formatted ASCII report for the current sequence and displays a Save dialog box that prompts you to save the report. The **Sequence Report...** button is visible at all operating levels. The default key assignment for **Sequence Report...** is <Ctrl-F4> (<command-F4>, <meta-F4>, <Alt-F4>).

Test Runtime Updates?



<Ctrl-F5>

When you select the **Test Runtime Updates?** checkbox, the Test Executive updates the Test Display each time a step finishes running. When you clear this checkbox, the Test Executive does not update the Test Display during sequence execution. To make the Test Executive run as fast as possible, disable Test Display updates by clearing this checkbox. The **Test Runtime Updates?** checkbox is visible at all operating levels. The default key assignment for **Test Runtime Updates?** is <Ctrl-F5> (<command-F5>, <meta-F5>, <Alt-F5>).

Operator Interface Key Assignments

To summarize, Table 3-1 lists the default key assignments for each operator interface control.

Table 3-1. Default Operator Interface Key Assignments

Control	Default Key Assignment
Open	<F2>
Close	<F3>
Login	<F4>
Edit	<F5>
New	<F6>
Quit	<F10>
Test UUT	<Shift-F1>
Single Pass	<Shift-F2>
Run Step(s)	<Shift-F3>
Loop Step(s)	<Shift-F4>
Stop On Any Failure	<Shift-F5>
Sequence Runtime Updates?	<Shift-F6>
Run Mode	<Shift-F7>
Abort Loop	<Shift-F9>
Abort	<Shift-F10>
Clear Step Status	<Ctrl-F1> (<command-F1>, <meta-F1>, <Alt-F1>)
Clear Test Display	<Ctrl-F2> (<command-F2>, <meta-F2>, <Alt-F2>)
View Test Report	<Ctrl-F3> (<command-F3>, <meta-F3>, <Alt-F3>)

Table 3-1. Default Operator Interface Key Assignments (Continued)

Control	Default Key Assignment
Sequence Report	<Ctrl-F4> (<command-F4>, <meta-F4>, <Alt-F4>)
Test Runtime Updates?	<Ctrl-F5> (<command-F5>, <meta-F5>, <Alt-F5>)

Indicators

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

Sequence Display

The Sequence Display is a control that contains two parts. The first is a ring control that allows the user to select the sequence currently visible in the second part, a multiple-selection list box. During sequence execution, the ring control also indicates the filename of the current sequence visible in the list box.

Each line in the list box portion of the display has three fields. From left to right, the fields are Run Mode, Step Name, and Step Status/Result. In addition, steps that are sequences are marked with the closed-folder glyph.



Note The fields in the Sequence Display are not labeled, so take special notice of what occurs in each field.

The Run Mode field indicates the setting of the **Run Mode** parameter for the step. Table 3-2 lists the possible **Run Mode** field values and their meanings.

Table 3-2. Run Mode Field Values

Value	Meaning
blank (no symbol)	Step runs normally.
S	Step is skipped.
P	Step is skipped with a forced PASS result.
F	Step is skipped with a forced FAIL result.

The Step Name field shows the name of the step.

The Step Status/Result field is set to `RUNNING` during the step execution to indicate the active step. After the step completes, the field is set to reflect the result of the step. Table 3-3 lists the possible Step Status/Result field values and their meanings.

Table 3-3. Step Status/Result Field Values

Value	Meaning
SKIP	Step did not execute.
PASS	Step result satisfied limit specification.
FAIL	Step result did not satisfy limit specification.
NONE	Step data was logged, but no comparison was made because the limit specification was set to <code>Log only</code> .
UNKNOWN	No step data was logged, and no comparison was made because the limit specification was left blank.
ERROR	Run-time error occurred during step execution.

Notice the difference between `NONE` and `UNKNOWN`. If the limit specification for a step is set to `Log only`, the Test Executive is instructed to log the step data but not make a comparison. The step result is `NONE` because there is no result. If the limit specification for a step is left blank, however, the Test Executive takes no action other than to run the step. Because there is no limit specification, the Test Executive has not been instructed on what to do with the step data, so the step result is `UNKNOWN`.

Test Display

The Test Display shows three types of information:

- Result of each step, if **Test Runtime Updates?** is enabled
- Error messages
- Test report

Result of Each Step

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

```

Step name      Result
                Comment (optional, might be multiple lines)
                Measurement      Comparison      Lower Limit
                Upper Limit or
                String Limit
                Execution Time      Time (in ms)
  
```

Notice that the number of lines that comprise the step result varies, depending on the type of comparison made and whether a step logs a comment. A step result always contains at least one line listing the name and result of the step. The result is the same value shown in the Step Status/Result field of the Sequence Display.

The format and content of the Comment line(s) is determined by the step that logged the comment. The Measurement line shows the measured value returned by the step. Comparison shows the type of limit checking used to determine if the step passed. Lower and Upper Limits are the numeric limit values used for PASS/FAIL determination. String Limit is the string value used for PASS/FAIL determination. Table 3-4 lists the possible values of Comparison and their relation to the limits (Condition for Step to Pass).

Table 3-4. Comparison Values and Relative Limits

Value	Condition for Step to Pass
EQ (==)	Numeric Measurement = Lower Limit
NE (!=)	Numeric Measurement != Lower Limit
GT (>)	Numeric Measurement > Lower Limit
LT (<)	Numeric Measurement < Lower Limit
GE (>=)	Numeric Measurement >= Lower Limit
LE (<=)	Numeric Measurement <= Lower Limit
GTLT (> && <)	Numeric Measurement > Lower Limit and < Upper Limit
GTLE (> && <=)	Numeric Measurement > Lower Limit and <= Upper Limit

Table 3-4. Comparison Values and Relative Limits (Continued)

Value	Condition for Step to Pass
GELT (>= && <)	Numeric Measurement >= Lower Limit and < Upper Limit
GELE (>= && <=)	Numeric Measurement >= Lower Limit and <= Upper Limit
STRCMP	String Measurement = String Limit

The Execution Time line shows the execution time of the step resource in milliseconds, if the step was executed.

Error Messages

Error messages also appear in the Test Display. One of two types of error messages can appear. The first is a parsing error. If a test sequence references an undefined resource or contains an invalid limit or dependency, a description of the error appears in the Test Display.

The second type of error message displays run-time errors. When the Test Executive detects a run-time error, it displays a description of the error in the Test Display.



Note The Test Executive engine does not log complete result information for steps when errors occur in a sequence. Therefore, the test report that the Test Executive generates will contain incomplete result information in the event of an error.

The Test Report

You also view the Test Report in the Test Display. The Test Report is a record of the testing results for the most recent execution of a test sequence. Figure 3-1 shows the format of a Test Report.

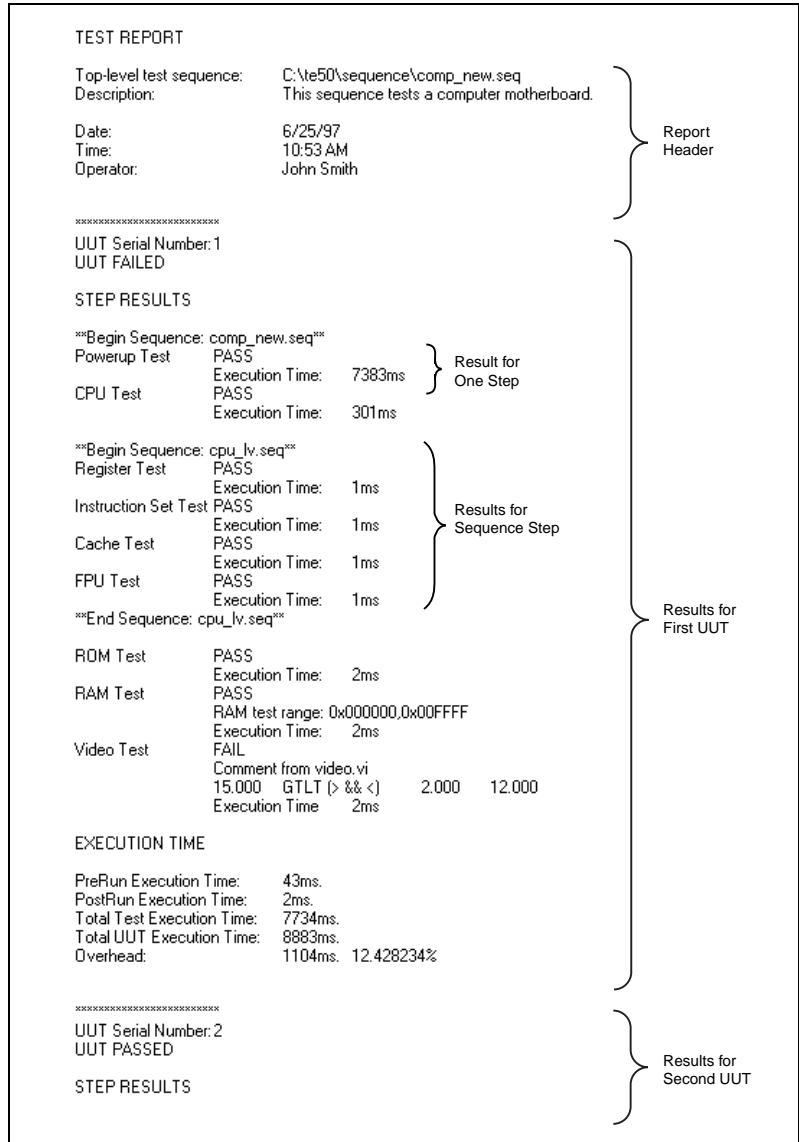


Figure 3-1. Sample Test Report

Status

The Status indicator appears directly above the Sequence Display. This indicator shows the current operating status of the Test Executive. Table 3-5 lists the possible values of the Status indicator and their meanings.

Table 3-5. Status Indicator Values

Value	Meaning
STOPPED	No test sequence currently running.
RUNNING	Test sequence is running.
LOOPING [n]	Test sequencing is looping on a step and is in the n th iteration.
QUIT	Test Executive has stopped executing.

Sequence Name

The Sequence Name indicator appears above the Sequence Information box. If a new sequence is currently visible in the Sequence Display, this indicator displays **Untitled sequence**. When a named sequence is displayed, the Sequence Name indicator displays the filename of the test sequence. During sequence execution, the Sequence Name indicator always displays the name of the top-level sequence.

Sequence Information

The Sequence Information indicator appears above the Test Display and below the Sequence Name. When a sequence is displayed, the Test Executive updates this box to display user-defined information. By default, the Sequence Information box displays the sequence description. During sequence execution, the Sequence Information indicator always displays the information for the top-level sequence.

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 action that each dialog box requires.

Default Login Dialog Box

The default Login dialog box prompts you to enter your name and password.

The password sets the operating level of the Test Executive. The login dialog box appears when the Test Executive first starts running and when you click the **Login** button. In the dialog box, use the <Tab> key to move between the Name and Password fields. Click the **OK** button or press <Enter> (<return>, <Return>) to confirm the entries made in the Name and Password inputs. Click the **Cancel** button to remove the dialog box without making any changes to the existing name and password. If you click the **Cancel** button when the Test Executive first starts running, the Test Executive runs at the Operator level.

For information about modifying the Default Login dialog box, refer to Chapter 5, *Modifying the Test Executive*.

Default Select Sequence Dialog Box

The default Select Sequence file dialog box prompts you to select a test sequence file to open.

The default Select Sequence file dialog box appears when you click the **Open** button on the operator interface panel. This dialog box is the standard LabVIEW file dialog box, which is initially configured to show only those files ending with an SEQ extension. If you select a valid test sequence file and click the **OK** button, the Test Executive opens the selected test sequence. If you click the **Cancel** button, the Test Executive closes the Select Sequence file dialog box and performs no operation.

For information about modifying the default Select Sequence file dialog box, refer to Chapter 5, *Modifying the Test Executive*.

Default UUT Information Dialog Box

The default 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.

The default UUT Information dialog box appears only when you click the **Test UUT** button. The Test Executive accepts any ASCII string as a valid serial number. Click the **OK** button or press <Enter> (<return>, <Return>) to confirm the serial number. Click the **Stop** button to stop UUT testing.

For information about modifying the default UUT Information dialog box, refer to Chapter 5, *Modifying the Test Executive*.

Default Test Failed Dialog Box

The default Test Failed dialog box appears when a step fails and the FAIL Action of the step is set to `Callback`. The dialog box prompts you to select an action—Continue, Stop, or Retry.

If you click the **Continue** button, the Test Executive logs the step failure and continues with the next step in the sequence. If you click **Stop**, the Test Executive stops testing the UUT. If you click the **Retry** button, the Test Executive runs the failed step again.

For information about modifying the default Test Failed dialog box, refer to Chapter 5, *Modifying the Test Executive*.

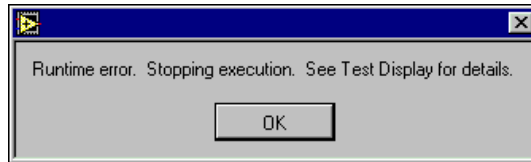
Default PASS/FAIL/ABORT Banners

The default PASS/FAIL/ABORT banners indicate whether the current UUT passed, failed, or was aborted. In Test UUT mode, one of these banners appears at the end of test sequence execution for each UUT. The default PASS/FAIL/ABORT banners do not appear in Single Pass mode. Click the **OK** button or press <Enter> (<return>, <Return>) to acknowledge a banner and continue testing.

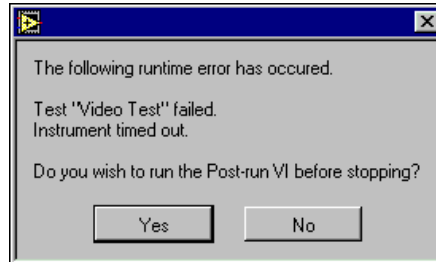
For information about modifying the PASS/FAIL/ABORT banners, refer to Chapter 5, *Modifying the Test Executive*.

Run-Time Error Warning Dialog Box

A Run-Time Error dialog box appears when a step reports an error. There are two types of Run-Time Error dialog boxes that appear. The following illustration shows the general Run-Time Error dialog box.



The dialog box shown in the following illustration appears when a run-time error occurs and you have specified a Post-Run VI for the test sequence. When this prompt appears, you can choose whether to run the Post-Run VI.



Parsing Error Dialog Box

The Parsing Error dialog box appears when the Test Executive detects that a test sequence contains one of the following errors:

- Invalid step resource
- Invalid step name
- Invalid step type
- Invalid limit specification
- Invalid dependency expression
- Invalid GOTO destination

After clicking the **OK** button in the Parsing Error dialog box, you can use the Sequence Errors dialog box to find and fix the errors in the test sequence. To show the Sequence Errors dialog box, select **Sequence»Sequence Errors...** in the Sequence Editor. For more details about the Sequence Errors dialog box, see Chapter 4, *Creating Tests and Test Sequences*.

Creating Tests and Test Sequences

This chapter describes the process of creating new test programs and test sequences for execution by the Test Executive. The Test Executive can call two different types of test programs—LabVIEW VIs and C functions (Windows NT/98/95 and UNIX).

Writing LabVIEW Tests

The LabVIEW VIs called by the Test Executive must have a specific front panel and connector pane structure. The Test Executive includes a wizard to help you create new Test VIs that meet this specification, or you can create test VIs manually.

To run the wizard, select **Project»Test Executive»Utilities»VI Wizard....** In the wizard, you can open template VIs for the various types of test VIs. The template VIs contain the correct front panels and connector panes for use with the Test Executive. To use a template VI, add your code to the block diagram and save the VI.

You also can create test VIs manually. Use the type definitions included with the Test Executive, which you can access by selecting **Controls»User Controls»Test Executive Typedefs**.

Every test VI must contain the Test Data cluster and error out cluster on its front panel and connector pane. The input buffer string and invocation info cluster are optional.



Note If the Test Executive calls a test VI that does not have the correct connector pane configuration, it attempts to assign the correct connector pane to the VI and then prompts you to save the VI with the new connector pane.






Required Indicators

Test Data Cluster

A test VI uses the Test Data cluster for transmitting result data needed by the Test Executive to make a PASS/FAIL determination.

Table 4-1 lists the elements contained in the Test Data cluster.

Table 4-1. Test Data Cluster Elements

Name	Type	Meaning
PASS/FAIL Flag		Set by test VI to indicate whether test passed or failed. This flag is used if the limit specification of the test is Boolean.
Numeric Measurement		Numeric measurement value used by Test Executive for Pass/Fail evaluation.
String Measurement		String value used by Test Executive for Pass/Fail evaluation.
User Output		String used to hold test-specific output data. Output data of any type can be flattened to string and passed out in this buffer.
Comment		Comment from VI that is included in the Test Report.

You must wire the Test Data cluster to the shaded connector pane terminal on your test VI, shown in the following illustration.






When you create a test VI, use the Test Executive typedef `TYPEDEF-Test Data.ct1` for result information. For more information about Test Executive typedefs, see Chapter 5, *Modifying the Test Executive*.

Error Cluster

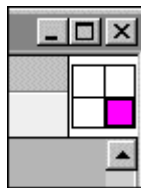
A test VI must contain an error out cluster. Table 4-2 lists the elements the error out cluster contains.

Table 4-2. Error Cluster Elements

Name	Type	Meaning
Status		True if an error occurred, False otherwise
Code		0 if no error, non-0 to indicate specific error
Source		Name of the VI that caused the error and text description of the error

The Test Executive uses the contents of the error out cluster to determine if a run-time error occurs and takes appropriate action, if necessary. When you create a test VI, use the standard LabVIEW error out cluster for error information.

You must wire the error out cluster to the shaded connector pane terminal on your test VI, shown in the following illustration.

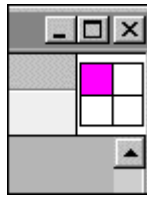


Optional Inputs

Input Buffer

A test VI optionally may have a string control on its front panel that serves as an input buffer. This input buffer allows your test sequence to specify input data for the test VI. The Test Executive does not define the meaning or content of the input buffer. It is a general-purpose mechanism for passing any data into a test VI. The test VI defines the meaning and content of this buffer. To add an Input Buffer control to your test VI, use the Test Executive typedef `TYPEDEF-Input Buffer.ct1`. For more information about Test Executive typedefs, see Chapter 5, [Modifying the Test Executive](#).

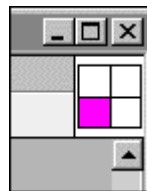
You must wire the Input Buffer to the shaded connector pane terminal on your test VI, shown in the following illustration.



Invocation Information

A test VI may have an Invocation Information cluster on its front panel to receive run-time calling information from the Test Executive. The Invocation Information cluster contains information about the sequence being run, the UUT being tested, and the name and loop count of the current step. To add an Invocation Information control to your test VI, use the Test Executive typedef `TYPEDEF-Invocation Information.ct1`. For more information about Test Executive typedefs, see Chapter 5, [Modifying the Test Executive](#).

You must wire the Invocation Information cluster to the shaded connector pane terminal on your test VI, shown in the following illustration.



Writing C Tests (Windows NT/98/95 and UNIX)

C-based tests that can be called by the Test Executive are functions in a 32-bit DLL (Windows NT/98/95) or shared library (UNIX) with a specific functional prototype. These test functions are equivalent to the functions called by the LabWindows/CVI Test Executive but can be created using any compiler capable of creating standard DLLs or shared libraries.

Test functions should have the following prototype:

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

Test Data Structure

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

```
typedef struct {
    int32  result;
    double measurement;
    char   *inBuffer;
    char   *outBuffer;
    char   *modPath;
    char   *modFile;
    void   *hook;
    int32  hookSize;
    void   *mallocFuncPtr;
    void   *freeFuncPtr;
} tTestData;
```

Table 4-3 lists the `tTestData` structure parameters.

Table 4-3. `tTestData` Structure Parameters

Name	Type	Description
result	32-bit signed integer	Set by test function to indicate whether test passed (result=1) or failed (result=0). This flag is observed only if the test has been set to pass or fail based on a Boolean comparison.
measurement	64-bit floating-point number	Measurement value used for Test Executive Pass/Fail evaluation.
inBuffer	string pointer	String passed in by the Test Executive.
outBuffer	string pointer	String passed out by the Test Executive.
modPath	string pointer	Directory path of file containing test function.
modFile	string pointer	Filename of file containing test function.
hook	generic pointer	Not used by the LabVIEW Test Executive.
hookSize	32-bit signed integer	Not used by the LabVIEW Test Executive.

Table 4-3. tTestData Structure Parameters (Continued)

Name	Type	Description
mallocFuncPtr	generic pointer	Function pointer to <code>malloc()</code> function used to allocate outBuffer and errorMessage in Test Error structure.
freeFuncPtr	generic pointer	Function pointer to <code>free()</code> function pointer to be used in conjunction with the <code>malloc()</code> function pointed to by mallocFuncPtr .

The Test Executive allocates and frees an input buffer when one is specified for the test. The test function must allocate the **outBuffer** if needed (using the `malloc()` function pointed to by **mallocFuncPtr**), but the Test Executive frees it. If your test function needs access to another file in its directory (such as a LabWindows/CVI `.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 containing the test.

Test Error Structure

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

```
typedef struct {
    int32 errorFlag;
    int32 errorLocation;
    int32 errorCode;
    char *errorMessage;
} tTestError;
```

Table 4-4 lists the `tTestError` structure parameters.

Table 4-4. tTestError Structure Parameters

Name	Type	Description
errorFlag	32-bit signed integer	errorFlag =1 if an error occurred, errorFlag =0 otherwise.
errorLocation	32-bit signed integer	Not used by the LabVIEW Test Executive.
errorCode	32-bit signed integer	errorCode =0 for no error, non-zero to indicate a specific error.
errorMessage	string pointer	Text description of error (allocated with <code>mallocFuncPtr</code>).

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

Compiling Test Functions

Test functions should be compiled using C calling conventions using any compiler that can produce standard 32-bit DLLs (Windows NT/98/95) or shared libraries (UNIX).

Creating Pre-Run and Post-Run VIs

Pre-run and Post-run VIs are special VIs that run at the beginning and end of each test sequence. They are used for test system configuration, such as turning on a vacuum pump or shutting down power supplies, as opposed to making measurements for a specific test. In general, the Pre-run and Post-run VIs always execute, regardless of the status of any test. If the Pre-run or Post-run VI encounters a situation that prevents the test sequence from executing, it indicates the condition as a run-time error. Pre-run and Post-run VIs do not log data. They simply return run-time error information.

Like standard LabVIEW tests, Pre-run and Post-run VIs must have the clusters Test Data and error out on their front panels and wired to the correct connector pane terminals.

What is a Test Sequence?

A test sequence consists of a collection of data that describes the flow of test execution. The main components of a test sequence are as follows:

- List of steps
- Pre-run VI (optional)
- Post-run VI (optional)
- Set of sequence callback VIs (optional)
- Dependencies for flow control based upon PASS/FAIL results
- Global flag for stopping testing on any failure
- Test report file information
- Description of the sequence
- Sequence load specification
- Sequence path specification

What is a Step?

A step consists of a combination of specifications that tell the Test Executive how to perform a single execution element in the testing process. Steps can be of four different types—LabVIEW Test, C Test, GOTO, or Sequence. For LabVIEW Test, C Test, or Sequence steps, the specifications are as follows:

- Name
- Resource file for LabVIEW VI, DLL/shared library, or Test Executive sequence file
- Function name for C Tests only
- Limit Specification for determining PASS/FAIL status of step for LabVIEW Tests and C Tests only
- Dependency expression
- Load specification indicating whether the step resource is pre-loaded or loaded dynamically
- Run Mode specifying whether step executes normally
- FAIL Action and maximum loop count if applicable
- Optional input specifiers (Input Buffer for LabVIEW Tests and C Tests only, Invocation Information for LabVIEW Tests only)
- Panel Display specifier to indicate whether the test VI should show its panel at run time for LabVIEW Tests only

For GOTO, the step specifications are as follows:

- GOTO target
- Dependency expression

Creating or Editing a Test Sequence

To create or edit a test sequence, you must invoke the Sequence Editor when the Test Executive is at the Developer operating level. This section describes the operation of the Sequence Editor.

Step Editing Elements

Insert

Use the **above** and **below** radio buttons to indicate where new steps are inserted or pasted into the sequence list. If you select the **above** button, new steps are inserted or pasted above the currently selected element in the sequence list. If you select the **below** button, new elements are inserted or pasted below the currently selected element. If you select more than one element in the sequence list, the Test Executive inserts or pastes the new elements either above or below the first or last element in your selection.

New Step

When you click the **New Step** button, the Sequence Editor inserts a new step into the sequence list. Use the **above** and **below** radio buttons to determine where the new step is inserted.

Copy Steps, Cut Steps, Delete Steps, Paste Steps, and Undo Step Edits

When you click the **Copy Steps** or **Cut Steps** button or select **Edit»Copy Steps** or **Edit»Cut Steps**, the Sequence Editor copies or cuts the currently selected sequence steps, including groups of steps, to the Sequence Editor clipboard.

When you click the **Delete Steps** button or select **Edit»Delete Steps**, the Sequence Editor deletes the currently selected steps.

When you click the **Paste Steps** button or select **Edit »Paste Steps**, the Sequence Editor inserts the contents of the Sequence Editor clipboard into the sequence list according to the setting of the Insert switch. When the Sequence Editor clipboard is empty, the **Paste Steps** button is disabled, and the **Edit»Paste Steps** menu item is dimmed.

When you click the **Undo Step Edits** button or select **Edit»Undo Step Edits**, the Sequence Editor reverses the last edit action. If you have not made any changes or have just reversed an action, the **Undo Step Edits** button is disabled, and the **Edit»Undo Step Edits** menu item is dimmed.



Note For a list of Element Editing Control key and menu shortcuts, see the [Sequence Editor Control Key Assignments](#) and [Sequence Editor Menu Shortcuts](#) sections in this chapter.

Using the Editing Elements

Adding a New Step

The Test Executive allows you to insert a new step either above or below an existing step in the sequence list. Perform the following operations to add a new step to a sequence.

1. Select the existing step in the sequence list and set the insert position to above or below, as desired. The insert position is set using the radio button control in the upper right corner of the Sequence Editor panel.
2. Click the **New Step** button. Notice that clicking this button adds a new, untitled step to the sequence list and selects the Name control, which allows you to name the new step.
3. Enter or select the desired values in the step editor controls. Notice that the Test Executive automatically applies the new values to the step.

Modifying a Step

Perform the following operations to modify a step.

1. Click the step you want to edit in the sequence list. Notice that the settings for the selected step appear in the step editing area.
2. Enter or select the desired values in the step editor controls. Notice that the Test Executive automatically applies the new values to the step.

Copying a Step

To copy a step, perform the following operations.

1. Click the step you want to copy in the sequence list.
2. Click the **Copy Steps** button or select **Edit»Copy Steps** to copy the selected step to the Sequence Editor clipboard.
3. To insert the step above an existing step, select the step and set the insert position to above. To insert the step below an existing step, select the step and set the insert position to below.
4. Click the **Paste Steps** button or select **Edit»Paste Steps**. The Sequence Editor inserts the contents of the clipboard in the desired location in the sequence list.

Deleting a Step

To delete a step, perform the following operations.

1. Click the step you want to delete in the sequence list.
2. Click the **Delete Steps** button or select **Edit>Delete Steps**.

Mass Editing

The Sequence Editor allows you to edit more than one step at a time. This feature is useful when you need to make the same modifications to several steps. To mass edit a group of steps, perform the following operations.

1. Select the steps you want to edit by <Shift>-clicking them in the sequence list.



Note To add or remove steps from the current selection, <Shift>-click the step. To add or remove a group of adjacent steps to the selection, hold down the <Shift> key, click the first step in the group, and drag the mouse pointer to the last step in the group.

2. Enter or select the desired values in the step editor controls. The Sequence Editor automatically applies the changes to every step in the selection.
3. To exit mass edit mode, select a single step in the sequence list.

Step Editor Controls

This section describes the step editor controls and indicators. You use these controls, such as Select Resource and the Set Limit Specification, to modify the definition of any step in the sequence.

Type

Use the Type ring control to choose the type of the selected step. The available step types are LabVIEW Test, C Test, GOTO, and Sequence. Depending on the type selected, other editor controls are shown or hidden.

Name (LabVIEW Test, C Test, Sequence)

Type an ASCII string in the Name control. The name appears in the Sequence Display of the main Test Executive front panel when the sequence is loaded. Each step in a sequence must have a unique name. The Sequence Editor warns you if a step is not properly named when you try to quit the editor.

Resource (LabVIEW Test, C Test, Sequence)

The Resource indicator contains the path to the resource that the step executes. For LabVIEW tests, the resource is a valid test VI. For C tests, the resource is a DLL or shared library containing a valid test function. For Sequence steps, the resource is a Test Executive sequence file. To set or change the resource, click the **Select Resource...** button. From the dialog

box that appears, select a valid resource file from the file system (either local or network). The path to the resource file you select appears in the Resource indicator.

Function (C Test)

Type the name of the function found with the DLL or shared library specified in the Resource indicator. This function must be a valid C test function.

Limit Specification (LabVIEW Test, C Test)

The Limit Specification indicator specifies the type of limit checking the Test Executive uses to determine if a step passes. You cannot type directly into the Limit Specification indicator. To specify a limit, click the **Set Limit Specification...** button next to the indicator. Then use the ring control to set the comparison type and the string controls to set the measurement.

The **Comparison Type** ring control specifies the type of comparison to perform, if any, to determine if a step passed. Select the desired limit type from the **Comparison Type** ring control. Table 4-5 lists the meaning of each value of Comparison Type.

Table 4-5. Comparison Type Values

Type	Condition for Test to Pass
EQ (==)	Numeric Measurement = Lower Limit
NE (!=)	Numeric Measurement != Lower Limit
GT (>)	Numeric Measurement > Lower Limit
LT (<)	Numeric Measurement < Lower Limit
GE (>=)	Numeric Measurement >= Lower Limit
LE (<=)	Numeric Measurement <= Lower Limit
GTLT (> && <)	Numeric Measurement > Lower Limit and < Upper Limit
GTLE (> && <=)	Numeric Measurement > Lower Limit and <= Upper Limit
GELT (>= && <)	Numeric Measurement >= Lower Limit and < Upper Limit
GELE (>= && <=)	Numeric Measurement >= Lower Limit and <= Upper Limit
Boolean	PASS/FAIL Flag = TRUE

Table 4-5. Comparison Type Values (Continued)

Type	Condition for Test to Pass
String	String Measurement = String Limit
Log only	No PASS/FAIL determination—measurement is logged
None	No PASS/FAIL determination or data logging

Depending on the setting of Comparison Type, zero, one-limit, or two-limit entry controls appear in the Set Limit Specification dialog box. A one-limit value appears for Comparison Types of EQ ($=$), NE (\neq), GT ($>$), LT ($<$), GE (\geq), or LE (\leq). Two-limit values, a lower and upper limit, appear for Comparison Types of GTLT ($> \&\& <$), GELE ($\geq \&\& <$), GELT ($\geq \&\& <$), or GTLE ($> \&\& \leq$).

When the Comparison Type is `String`, the string limit control appears. The Test Executive compares the string value with the string measurement that the step calculates. There are no limits for Comparison Types of `Boolean`, `Log only`, or `None`.

For numeric comparisons, use the Format control to view the numeric limits in fractional, scientific, decimal, hexadecimal, octal, or binary format. For string comparisons, use the Display control to view the string limit in normal, hex, or ‘\’ codes mode.

Load Specification (LabVIEW Test, C Test, Sequence)

Load Specification determines whether the step resource loads when the sequence loads (*pre-load*) or loads upon demand when the step executes (*dynamic-load*). Pre-load steps run much faster because the Test Executive does not need to load them during the test sequence. The Test Executive loads dynamic-load steps just before running them and unloads them immediately after running them.

Run Mode (LabVIEW Test, C Test, Sequence)

Run Mode specifies how the step executes. Table 4-6 lists the options for Run Mode and their meanings.

Table 4-6. Run Mode Options

Run Mode	Meaning
Normal	Execute step normally.
Skip	Do not execute the step; set result to <code>SKIP</code> .
Force PASS	Do not execute the step; set result to <code>PASS</code> .
Force FAIL	Do not execute the step; set result to <code>FAIL</code> .

FAIL Action (LabVIEW Test, C Test, Sequence)

FAIL Action specifies an action to take if the step fails. Table 4-7 lists the options for FAIL Action and their meanings.

Table 4-7. FAIL Action Options

FAIL Action	Meaning
Continue	Continue execution with next step.
Stop	Stop execution of sequence.
Callback	Call the Test Failure sequence callback VI to determine whether to continue, stop, or retry the step.
Loop	Repeat execution of the step.

Max Loop Count

The Max Loop Count control appears only when FAIL Action is set to `Loop`. Max Loop Count specifies the maximum number of loop iterations to perform when the FAIL Action is set to `Loop`, and the step fails. Setting the loop count to `-1` causes the step to loop until the operator clicks on either the **Abort Loop** or **Abort** button, or the step passes.

Input Buffer? (LabVIEW Test, C Test)

The Input Buffer? control specifies whether the Test Executive should pass a buffer of input data to the test VI or function when it executes the step. If you clear this checkbox, the Test Executive does not pass any input data. If

you check this box, the Input Buffer string control appears, which allows you to enter the input data. To pass input data to a test VI, the test VI must have an Input Buffer string control on its front panel. Otherwise, the Test Executive cannot execute the step. For more details about the Input Buffer? control, see the [Test Executive Typedef Controls](#) section in Chapter 5, [Modifying the Test Executive](#). No additional operations are necessary to pass input data to a C test function.

Invocation Info? (LabVIEW Test)

The Invocation Info? control specifies whether the Test Executive passes run-time call information to the test VI when it executes the step. If you clear this checkbox, the Test Executive does not pass run-time call information to the test VI. If you check this box, the Invocation Info? control appears, which allows you to enter the run-time call information. To pass invocation information to a test VI, the test VI must have an Invocation Information cluster control on its front panel. Otherwise, the Test Executive cannot execute the step. For more information about the Invocation Info? control, see the [Test Executive Typedef Controls](#) section in Chapter 5, [Modifying the Test Executive](#).

Show Test VI Panel at Runtime? (LabVIEW Test)

The Show Test VI Panel at Runtime? control determines whether the test VI shows its panel at run time. If you select this checkbox, the test VI panel shows at run time. Otherwise, the test VI panel remains closed.

Edit Test VI (LabVIEW Test)

The **Edit Test VI** button is linked to a callback VI. The default Open Test VI callback opens the test VI panel for the current step. If the test VI is not defined or is invalid, the callback VI cannot open the panel and displays an error message. You use the **Edit Test VI** button to edit test VIs from within the Sequence Editor. Modifying or replacing the Open Test VI callback VI makes the **Edit Test VI** button perform different functions. For more details, see the [Advanced Modifications](#) section in Chapter 5, [Modifying the Test Executive](#).

Edit Dependencies

The **Edit Dependencies...** button invokes the Dependency Editor, allowing you to examine or modify the dependencies for the current step. For more details, see the [Editing Dependencies](#) section of this chapter.

Edit Step Comment (LabVIEW Test, C Test, GOTO, Sequence)

With the **Edit Step Comment...** button, you can edit the comment field for the step. The Sequence Report callback VI can access this comment. If you click the **Edit Step Comment...** button, the Edit Step Comment dialog box appears.

GOTO Target (GOTO)

Enter the name of a step in the GOTO Target control. If the Test Executive executes this step, sequence execution skips to the GOTO Target. Within the Test Executive, step names are case-sensitive.

GOTO Conditions (GOTO)

The **GOTO Conditions...** button invokes the Dependency Editor, allowing you to examine or modify the dependencies for the current GOTO step. For more information, see the [Editing Dependencies](#) section later in this chapter.

Sequence Options

To view or set the Sequence Options, select **Sequence»Sequence Options...**

Sequence Load Specification

You use the Sequence Load Specification control to override the individual load settings for every step in the sequence. The default setting is *Use each step's load specification*, which means that when the Test Executive loads the sequence, it uses the load specification of each step to determine whether to pre-load or dynamically load step resources. If you set the Sequence Load Specification to *Pre-load all steps*, the Test Executive pre-loads every step resource in the sequence regardless of the load specification of the step. If you set the Sequence Load Specification to *Dynamic-load all steps*, the Test Executive dynamically calls each step resource, regardless of the load specification of the step. When all step resources are pre-loaded, the Test Executive runs faster, but it also uses more memory. When step resources are dynamically loaded, the Test Executive runs more slowly but uses less memory.

Sequence Path Specification

To make moving sequence files between computers easier, you can specify how the Test Executive resolves paths to step resources, sequence VIs, and the test report file specified in a sequence file. Through the Sequence Options dialog box in the Sequence Editor, you can set the Sequence Path Specification to `Absolute`, `Relative to sequence file`, or `Relative to Test Executive` default.

With a Sequence Path Specification of `Absolute`, the Test Executive expects to find step resources, sequence VIs, and the test report file in the exact location where they were when you saved the sequence file.

For a Sequence Path Specification of `Relative to sequence file`, the Test Executive expects to find test resources, sequence VIs, and the test report file in positions relative to the location of the sequence file.

When the Sequence Path Specification is set to `Relative to Test Executive` default, the Test Executive expects to find test resources, sequence VIs, and the step report file in positions relative to the path specified in the `DefaultResourceFilePath` preference in the `testexec.ini` file. For additional information about the `testexec.ini` configuration file, see the *System Configuration File, testexec.ini* section in Chapter 5, *Modifying the Test Executive*.

Stop on Any Failure

If set, the **Stop on Any Failure** checkbox stops sequence execution immediately after any step fails. This setting overrides the FAIL Action of any individual step.

Description

The **Description** button allows you to edit the description of the test sequence. If you click the **Description** button, the Edit Description dialog box appears. The description you create in this dialog box appears in the Test Report that the Test Executive generates at the end of test sequence execution.

Enable Test Report Logging

The Test Executive logs the Test Report to the file specified, if any, in the Test Report File indicator.

Report File Mode

The Report File Mode control specifies the Test Executive response when a file already exists with the name in the Test Report File indicator. If you set Report File Mode to **Overwrite**, the new report replaces any existing file contents. If you set Report File Mode to **Append**, the Test Executive appends the new report to the existing file. If the file does not exist, the Test Executive creates a file, regardless of the mode.

Change Report File

The **Change Report File** button specifies the pathname for the ASCII Test Report file generated at the end of test sequence execution. The path to the file you specify appears in the Test Report File indicator.

Sequence VIs

The term sequence VIs refers to the Pre-Run VI, Post-Run VI, and sequence callback VIs for a particular sequence.

You use the Select VI control to choose any one of the sequence VIs. Then, you change the path for that VI in the VI Path control. Alternately, clicking the **Browse** button activates a file dialog box, so you can browse for a new sequence VI. If you select a new sequence VI and click the **OK** button, the path to that VI is stored in the VI Path control.

To open the front panel of the current sequence VI, click the **Open VI** button. You can then examine or edit the VI. If the VI Path for the current sequence VI is empty or invalid, the Sequence Editor is not able to open the panel and displays an error message.

To clear the current sequence VI, click the **Clear VI** button. To save the changes that you make in the Sequence Options dialog box, click the **OK** button. To discard the changes, click the **Cancel** button.

Sequence Errors

When you select **Sequence»Sequence Errors...**, the Sequence Editor checks the test sequence for errors. If the Sequence Editor finds no errors in the sequence, it displays the message **No Sequence Errors**. If it finds errors in the sequence, the Sequence Editor displays the Sequence Errors dialog box.

The Sequence Errors dialog box lists steps that have errors in the **Bad Steps** list box. When you select a step from the **Bad Steps** list box, the errors for the selected step appear in the **Step Problems** list box. If you select an error

in the **Step Problems** list box, an explanation of the selected error appears in the **Details** list box.

For some errors, the Sequence Errors dialog box displays a Corrective Action. For example, if a step has an empty name, the Sequence Errors dialog box displays a Corrective Action of changing the name of the step. However, if the error is that the step does not have a resource, the Sequence Errors dialog box does not display a Corrective Action. Instead, it suggests that you assign a resource to the step in the Sequence Editor. Table 4-8 lists the errors that the Sequence Editor detects and the corresponding Corrective Actions.

Table 4-8. Possible Errors and Corrective Actions in the Sequence Errors Dialog Box

Error	Corrective Action
Step with no name	Change selected step name
Step with duplicate name	Change selected step name
Step with invalid resource	None
Step with invalid limit specification	None
GOTO step with invalid target	Change GOTO target in selected GOTO or all GOTOs
Step with error in dependency	None
Step dependent on invalid step	Change step name in selected dependency, selected expression, or all expressions
Step with invalid dependency condition	None

If you enter a Corrective Action, you must click the **Apply** button to apply the corrective action to the sequence. After you click the **Apply** button, the Sequence Errors dialog box updates to reflect the application of the Corrective Action. When you fix the last error in the sequence, the message **There are no sequence errors.** appears in the **Details** list box.

Clicking the **OK** button in the Sequence Errors dialog box returns you to the Sequence Editor and updates any changes you make. Clicking the **Cancel** button returns you to the Sequence Editor and discards any changes you make.

File Menu

Selecting **File»Save...** saves the current sequence. If the sequence has not yet been saved, a dialog box prompts you for a path or filename. Selecting **Save As...** prompts you for a filename and then saves the current sequence under that name. Selecting **Exit** returns you to the main Test Executive front panel, prompting you to save if any changes were made. If you made changes to the sequence or saved it under a different name, the Test Executive unloads the old sequence hierarchy and reloads the new, saved hierarchy.

When you select **Save...** or **Save As...**, the Sequence Editor scans the test sequence for errors. If any errors are found, you are prompted to fix them. If you choose to fix the sequence errors, the Sequence Error dialog box appears. Selecting **Save...** or **Save As...** without fixing sequence errors saves a sequence that cannot be executed by the Test Executive until the errors are fixed.

Edit Menu

Use the **Cut Steps**, **Copy Steps**, **Paste Steps**, **Delete Steps**, and **Undo Step Edits** menu items to edit steps as described in the [Step Editing Elements](#) section earlier in this chapter.

Use the **Cut**, **Copy**, **Paste**, **Clear**, and **Undo** menu items to edit data in Sequence Editor controls other than the Steps List. For example, if you edit the contents of the Input Buffer string, use the menu items listed in this paragraph.

Sequence Editor Control Key Assignments

Use the keyboard shortcuts listed in Table 4-9 for the Sequence Editor controls.

Table 4-9. Key Assignments for Sequence Editor Controls

Control	Key Assignment
Sequence List	<F3>
Insert	<F4>
New Step	<F5>
Copy Steps	<F6>
Cut Steps	<F7>

Table 4-9. Key Assignments for Sequence Editor Controls (Continued)

Control	Key Assignment
Delete Steps	<F8>
Paste Steps	<F9>
Undo Step Edits	<F10>
Type	<Shift-F1>
Name	<Shift-F2>
Function	<Shift-F3>
Set Limit Specification...	<Shift-F4>
Select Resource...	<Shift-F5>
Edit Dependencies	<Shift-F6>
Edit Step Comment	<Shift-F7>
Edit Test VI	<Shift-F8>
GOTO Target	<Shift-F9>
GOTO Condition	<Shift-F10>
Load Specification	<Ctrl-F1>
Run Mode	<Ctrl-F2> (<command-F2>, <meta-F2>, <Alt-F2>)
FAIL Action	<Ctrl-F3> (<command-F3>, <meta-F3>, <Alt-F3>)
Max Loop Count	<Ctrl-F4> (<command-F4>, <meta-F4>, <Alt-F4>)
Input buffer?	<Ctrl-F5>
Invocation Info?	<Ctrl-F6>
Show VI Panel at Runtime?	<Ctrl-F7>
Input Buffer	<Ctrl-F8>

Sequence Editor Menu Shortcuts

Use the keyboard shortcuts listed in Table 4-10 for the Sequence Editor menu commands.

Table 4-10. Sequence Editor Menu Commands

Menu Item	Key Assignment
File»Save	<Ctrl-S> (<command-S>, <meta-S>, <alt-S>)
Undo Step Edits	<Ctrl-Shift-Z> (<command-Shift-Z>, <meta-Shift-Z>, <alt-shift-Z>)
Cut Steps	<Ctrl-Shift-X> (<command-Shift-X>, <meta-Shift-X>, <alt-shift-X>)
Copy Steps	<Ctrl-Shift-C> (<command-Shift-C>, <meta-Shift-C>, <alt-shift-C>)
Paste Steps	<Ctrl-Shift-V> (<command-Shift-V>, <meta-Shift-V>, <alt-shift-V>)
Undo	<Ctrl-Z> (<command-Z>, <meta-Z>, <alt-Z>)
Cut	<Ctrl-X> (<command-X>, <meta-X>, <alt-X>)
Copy	<Ctrl-C> (<command-C>, <meta-C>, <alt-C>)
Paste	<Ctrl-V> (<command-V>, <meta-V>, <alt-V>)

Editing Dependencies

Conditional execution of one step based on the result of another is called a *dependency*. Use the Dependency Editor dialog box to define dependencies for steps.

The Dependency Editor dialog box shows the dependencies for all steps in the sequence. The name of each step appears in the **Steps** list box in the top-left corner of the Dependency Editor dialog box. To see the

dependencies for any step in the sequence, select that step in the **Steps** list box. The Dependency Editor then displays the dependencies for that step in the **Dependencies** list box. At the same time, the Dependency Editor updates the **New Determinants** list box to show the list of steps that can be added to the dependencies. If you select a GOTO step in the **Steps** list box, the **New Determinants** list includes every step in the sequence. If you select a step, the **New Determinants** list includes every step in the sequence except for that step. A step cannot be dependent on itself.

Perform the following operations to make a step dependent on the outcome of another step.

1. Select the dependent step in the **Steps** list box.
2. Select the determinant step in the **New Determinants** list box and click the » button to add it to the **Dependencies** list box. This adds a FAIL dependency for the determinant step to the **Dependencies** list box. Double-clicking the determinant step also adds the dependency. The FAIL dependency means that the dependent step executes only if the determinant step fails.
3. To change the dependency to a PASS dependency, click the **Change to PASS** button to the right of the **Dependencies** list box. Double-clicking the FAIL dependency also changes it to a PASS dependency.

Notice the Dependency Expression indicator at the bottom of the Dependency Editor dialog box. This indicator displays the same information as the **Dependencies** list box but in a different notation. The Test Executive uses this notation to store the dependencies for any given step.



Tip It is possible to make step A dependent on the outcome of step B even if step B comes after step A in the sequence list. Through the use of GOTOS, step B can execute before step A. If step B does not execute before step A, the result for step B is UNKNOWN. In such a case, any PASS/FAIL dependency that step A has upon step B evaluates to FALSE.

AND and OR Expressions

When a step has more than one determinant step, an AND or an OR expression must define the relationship between the determinants. For example, suppose Test C is dependent on Tests A and B. If Test C is dependent upon Test A passing and Test B failing, then Test C has an AND dependency expression on Tests A and B. If Test C is dependent upon Test A passing or Test B failing, then Test C has an OR dependency expression on Tests A and B.

Perform the following operations to add an AND dependency expression for a step.

1. Select the desired dependent step in the **Steps** list box.
2. Click the **Insert AND** button to the right of the **Dependencies** list box to add a new, empty AND expression. Click the `BEGIN AND` statement and set the Insert switch to below.
3. Add the desired determinant steps as described in the previous section.

When the Test Executive evaluates the AND expression, it evaluates each element between the `BEGIN AND` and `END AND` statements. The AND expression only evaluates TRUE if every element inside it is TRUE.

OR expressions are similar to AND expressions in that they contain several elements within a `BEGIN OR` and an `END OR` statement. An OR expression, however, evaluates TRUE if any one of the elements inside it is TRUE. You can add a new OR expression to the dependencies for any step or GOTO statement by clicking the **Insert OR** button. You add determinant steps to the OR expression in the same way that you add them to an AND expression.

You can change any AND expression to an OR expression, or an OR expression to an AND expression, by selecting the `BEGIN OR` or `END OR` statement of the expression and clicking the **Change to PASS** button. You can also change an expression by double-clicking its `BEGIN OR` or `END OR` statement.

Complex Dependencies

You can create complex dependency expressions by nesting AND and OR expressions inside each other. For example, suppose that Test D should execute only if Test A passes or if Tests B and C fail. To define such a dependency in the Dependency Editor, perform the following operations.

1. Select `Test D` in the **Steps** list box.
2. Click the **Insert OR** button. After inserting the OR expression, select the `BEGIN OR` statement and set the Insert switch to c.
3. Select `Test A` in the **New Determinants** list box and add it to the **Dependencies** list box. Use the **Change to PASS** button or double-click the FAIL dependency to change it to a PASS dependency.
4. Click the **Insert AND** button to nest an AND expression inside the OR expression. After inserting the AND expression, click the `BEGIN AND` statement and set the Insert switch to below.
5. Add FAIL dependencies for Tests B and C by selecting them in the **New Determinants** list box and clicking the » button.

Copy, Cut, Delete, Paste, and Undo

The Dependency Editor dialog box features cut, copy, paste, delete, and undo capabilities. You cut, copy, or delete statements from the dependencies of any step by selecting the statements in the **Dependencies** list box and clicking the **Copy**, **Cut**, or **Delete** buttons. You paste statements into the dependencies of any other step by clicking the **Paste** button. When the Dependency Editor clipboard is empty, the **Paste** button is disabled.



Note You can copy, cut, or delete individual statements or entire AND and OR expressions in the Dependency Editor. However, you cannot cut, copy, or delete partial expressions. When you select a partial expression, the **Copy**, **Cut**, and **Delete** buttons are disabled.

The Dependency Editor dialog box features one level of undo. You reverse any edit action by pressing the **Undo** button. When you have not made changes or have just reversed an action, the **Undo** button is disabled.

Dependency Editing Rules

You cannot insert or paste elements outside the top-level dependency expression in the Dependency Editor. If you attempt to insert or paste elements outside the top-level expression, the Dependency Editor inserts the elements inside the expression.

OK

The **OK** button saves any changes you make to the sequence dependencies and returns you to the Sequence Editor. Clicking the **OK** button saves only to memory. You must select **File» Save** or **File» Save As...** in the Sequence Editor to save changes to disk.

Cancel

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

Dependency Editor Key Assignments

Table 4-11 lists the Dependency Editor controls and their keyboard shortcuts.

Table 4-11. Dependency Editor Key Assignments

Control	Key Assignment
Sequence Elements	<F2>
New Determinants	<F3>
»	<F4>
Dependencies	<F5>
Insert	<F6>
Insert OR	<Shift-F1>
Insert AND	<Shift-F2>
Copy	<Shift-F3>
Cut	<Shift-F4>
Delete	<Shift-F5>
Paste	<Shift-F6>
Undo	<Shift-F7>
Change to PASS	<Shift-F8>

Relationship among Dependencies, Run Mode, and Test Flow

The dependencies and run mode for each step determine the flow of execution for a test sequence. The Test Executive performs the following steps to determine whether to execute a given step.

1. The Test Executive evaluates the dependencies for the step. For a step to execute, the dependency expression for that step must evaluate to TRUE. If the Test Executive determines that the current step should be skipped, the step result is set to SKIP.
2. If the dependencies indicate that the step should execute, the Test Executive evaluates the Run Mode of the step. If the Run Mode is Normal, the step executes. If Run Mode is set to any value other than

Normal, the step does not execute. Table 4-12 shows the corresponding step results for each Run Mode value.

Table 4-12. Run Mode Step Result Values

Run Mode	Test Result
Skip	SKIP
Normal	PASS/FAIL
Force PASS	PASS
Force FAIL	FAIL

When evaluating dependencies, the Test Executive does not distinguish between a real PASS/FAIL result—where the step actually executed—and a forced PASS/FAIL result.

Modifying the Test Executive

This chapter describes the architecture of the Test Executive and explains how to make modifications to it. If you do not plan to modify the Test Executive, you can skip this chapter. The chapter covers the following topics:

- System configuration file, `testexec.ini`
- Operator interface VI
- Callback VIs
- Test Executive typedef controls
- Common modifications
- Advanced modifications

System Configuration File, `testexec.ini`

The system configuration file, `testexec.ini`, is an ASCII file that defines the names and locations of the Test Executive operator interface VI, callback VIs, and preference values. The `testexec.ini` file is located in the Test Executive installation directory. The Test Executive creates a default configuration file with paths to the default operator interface VI, default callback VIs, and initial preference values if one does not exist when it starts running.

The system configuration file is divided into three sections. The first section, `[Callback Paths]`, identifies the locations of all default callback VIs. The second section, `[Operator Interface Path]`, identifies the location of the default operator interface VI. The third section, `[Preferences]`, lists the preference values that the Test Executive uses.

[Callback Paths] Section

Each time you launch the Test Executive, it loads the appropriate callback VIs by reading the `[Callback Paths]` section of the `testexec.ini` file. The entries in this section have the following format:

```
VI_id="VI_path"
```

VI_id specifies a Test Executive callback VI. The system configuration file must have an entry for each of the following VI_id string values:

- Login
- Select_Sequence
- Open_Sequence
- Save_Sequence
- Close_Sequence
- Exit
- Sequence_Report
- Default_PreUUT_Loop
- Default_PreUUT
- Default_PostUUT_Loop
- Default_PostUUT
- Default_PreStep
- Default_PostStep
- Default_Test_Report
- Default_Post_Run-Loop_Test
- Default_Test_Failure
- Default_Edit_Test_VI

VI_path specifies the absolute file path to the callback VI in a platform independent format. You must enclose VI_path in double quotes and make sure that it contains no extra leading or trailing spaces. The following code shows a sample entry in `testexec.ini` for the Login callback VI:

```
Login="/C/LV51/LVEXEC511/CALLBACK.LLB/Login
Callback.vi"
```

Patching Callback Paths

As described in the previous section, when you launch the Test Executive, it loads the appropriate callback VIs by reading the paths from the `testexec.ini` file. If the path to a particular callback VI is invalid, the Test Executive prompts you to find the callback VI. This happens, for example, if you deploy the Test Executive to a different machine and do not update the callback paths in the system configuration file.

After you locate the missing callback VI, the Test Executive patches the invalid callback VI path with the new path. Then, it prompts you to add the

selected VI library or directory to the search path. If you click the **Yes** button, the Test Executive automatically searches that VI library or directory to patch the missing paths for any other callback VIs. This search path is a temporary path that is deleted after you launch the Test Executive.



Note When the Test Executive patches a callback VI path, it marks the path as modified. If you try to close the Test Executive with patched callback VI paths, the Test Executive prompts you to save changes to `testexec.ini`. If you click the **Yes** button, the Test Executive saves the patched callback VI paths to `testexec.ini`. The next time you launch the Test Executive, it does not need to search for the callback VIs.

[Operator Interface Path] Section

When you launch the Test Executive Development System from the LabVIEW **Project** menu, the Test Executive loads the appropriate operator interface VI by first reading the [Operator Interface Path] section of the `testexec.ini` file. The entries in this section have the same format as those in the [Callback Paths] section. The system configuration file must have an entry for the following `VI_id` string value:

```
Operator_Interface
```

[Preferences] Section

The [Preferences] section of the `testexec.ini` file holds additional information the Test Executive uses to define certain values. The entries in this section have the following format:

```
preference_name=value
```

`preference_name` specifies the particular preference.

The first preference the default `testexec.ini` file specifies is `TestNameDisplayLength`, which is set to a default value of 21. The `TestNameDisplayLength` preference specifies how long the step names shown in the Test Executive Sequence Display can be before they are truncated. If you would like to display longer step names, increase the value of this preference in `testexec.ini` and expand the Sequence Display list box to accommodate the longer names.

The second preference the default `testexec.ini` file specifies is `DefaultResourceFilePath`, which has a default value of your Test Executive installation directory. When the Sequence Path Specification is set to `Relative to Test Executive default`, the Test Executive uses the value of this preference as the Test Executive default directory.

Operator Interface VI

The operator interface VI is the main panel of the Test Executive. It is responsible for accepting commands from the operator and passing them to the Test Executive engine. It also is responsible for receiving information from the Test Executive engine and displaying that information to the user.

The Test Executive package includes a default operator interface VI, called `Test Executive`. It is installed in the `OPERATOR.LLB VI Library` in the Test Executive installation directory. The Test Executive also allows you to customize the name, appearance, and/or behavior of the default operator interface VI.

Modifying the Default VI

Before you modify the default operator interface VI, make a backup copy of it.

Front Panel

You can open and examine the front panel of the default operator interface VI by performing the following steps.

1. Launch LabVIEW.
2. Select **File»Open** from the menu on the front panel and choose the Test Executive VI in the `OPERATOR.LLB VI Library` in the Test Executive installation directory. A Login dialog box appears because the default operator interface VI is configured to run when opened.
3. Log in as a developer and quit the Test Executive.

In addition to the visible controls and indicators, the front panel also contains some transparent string indicators. Dashed outlines mark the locations of the transparent string indicators.

You can make cosmetic modifications to the operator interface VI front panel, such as resizing buttons, changing function key assignments, or pasting in a logo or other imported graphics, without editing the block diagram. However, to add new controls to the panel or change the behavior of existing controls, you must edit the block diagram.

Block Diagram

To open and examine the block diagram of the operator interface VI, select **Windows»Show Diagram** from the menu on the front panel.

Notice that the operator interface diagram consists of a command loop and a clock loop. When you run the operator interface VI, it simultaneously runs the command loop and the clock loop until you quit the Test Executive. Notice also that the operator interface VI communicates with the Test Executive engine through subVI calls. The Test Executive allows you to open the front panel of any of these subVIs, but you cannot modify them or look at the block diagrams. The engine subVIs are shipped without block diagrams.

Command Loop

The command loop of the operator interface VI is a state machine construct described in the Spring 1996 issue of the *LabVIEW Technical Resource*. Refer to the *About this Manual* chapter, of this manual, for *LabVIEW Technical Resource* Spring 1996 issue ordering information. It continuously scans all active controls on the operator interface panel for changes in state. When a control changes state, the corresponding case in the command loop executes.

If you add a new control to the front panel of the operator interface, you must edit the command loop so that it monitors and handles state changes in the new control. Perform the following steps to edit the command loop.

1. At the left side of the command loop diagram and in the No Event Case structure, turn the state change of your new control into a Boolean value—TRUE if the state has changed and FALSE if it has not. The default command loop contains examples of Boolean controls and numeric controls.
2. In the No Event Case structure, add an input to the bottom of the Build Array function and wire the state change value into the new input.
3. On the front panel of the operator interface, add an item to the Event List enumerated type representing your new event. Always add new events immediately before the Update Display event. Set the Event List back to No Event after adding the new event.
4. Add a case before the Update Display case in the Command Handler case structure and place the code to handle your new command in the new case.



Note If you create a new subVI and call it from your operator interface block diagram, store the new subVI in `OPERATOR.LLB`, so the operator interface VI finds the subVI when it is loaded.

Callback VIs

When you install the Test Executive, the installer places a complete set of default callback VIs in the `CALLBACK.LLB` VI Library in the Test Executive installation directory. In addition, it installs a set of typedef controls for creating your own callback VIs in the `LabVIEW USER.LIB` directory.

By creating your own callback VIs, you customize certain operations in the Test Executive, such as user login, prompting for UUT information, displaying PASS/FAIL banners, logging UUT Test results, and generating Test and Sequence reports. The Test Executive engine then handles these operations by calling your callback VIs.

The callback VIs for different operations have different calling interfaces. The calling interface specifies a set of required inputs and outputs for the front panel of the callback VI. For the Test Executive to successfully call a callback VI, the callback VI must have all required inputs and outputs, and the inputs and outputs must be wired to the correct terminals on the connector pane of the VI. Refer to the example callback VIs for the required connector pane configuration for each callback VI.



Note If the Test Executive calls a callback VI that does not have the correct connector pane configuration, it attempts to assign the correct connector pane to the VI and prompts you to save the VI with the new connector pane.

You use the typedef controls for the correct definitions for some of the required callback inputs and outputs. These controls are available from the **Controls** palette after you install the Test Executive development system. For a detailed examination of these typedef controls, see the [Test Executive Typedef Controls](#) section later in this chapter.

Test Executive Callback VI Calling Interface

The Test Executive calling interface includes system callback VIs and sequence callback VIs.

System Callbacks

The Test Executive includes the following system callback VIs:

- Login
- Select Sequence
- Open Sequence

- Close Sequence
- Save Sequence
- Sequence Report
- Exit

The system callback VIs are not associated with the execution of any particular sequence. Each time you launch the Test Executive, it loads the appropriate system callback VIs by reading the entries in the system configuration file, `testexec.ini`. If the Test Executive cannot find the system configuration file, it prompts you to find it. When you quit the Test Executive, it unloads the system callback VIs.

The rest of this section contains a detailed description of the calling interface for each system callback VI.



Note In the parameter tables that follow, the type of some inputs and outputs is listed as `TYPEDEF-xxx.ct1`. This value indicates that the input or output is a Test Executive typedef control. For a detailed examination of these typedef controls, see the [Test Executive Typedef Controls](#) section of this chapter.

Login

The Test Executive calls the Login callback VI upon startup and whenever the user clicks the **Login** button on the operator interface panel. The Test Executive installs the default `Login Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.
Output	New Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the new user.

The default Login callback VI performs no action with its inputs. It sets a warning in the error out control if the user cancels the Login dialog box. If the login is successful, the default callback VI sets error out to `no error` and puts the name, password, and privilege level (Developer, Technician, or

Operator) of the new user into the New Login Info control. If the Login callback VI returns an error or warning, the Test Executive does not change the current login information and displays the error or warning in the Test Display.

You customize the Login callback VI to work with custom login information in the following manner. You first create a custom Login Info typedef to contain the custom login information. Then, when the login callback VI logs in a new user, it fills in the custom Login Info typedef with the custom login information, flattens the contents of the typedef to a string, and passes the flattened string out in the User Info output of the New Login Info cluster. If any other callback VI needs to access this custom login information, it uses the Unflatten From String function on the custom Login Info typedef.

Select Sequence

The Test Executive calls the Select Sequence callback VI when the operator clicks the **Open** button on the main operator interface panel to open a new test sequence. With the Select Sequence callback VI, the operator chooses the path to the new test sequence file. The Test Executive installs the default `Select Sequence Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Current Sequence Info	TYPEDEF-Sequence.ct1	Contains information about the test sequence currently shown in the Sequence Display.
Input	Current Login Info	TYPEDEF-Login Info.ct1	Contains login information for the current user.
Output	Sequence Path	String	Contains the path to the selected test sequence file.
Output	Cancelled?	Boolean	This output is TRUE if the operator cancelled the Select Sequence operation, FALSE otherwise.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Select Sequence callback VI opens the standard LabVIEW file dialog box and prompts the operator to choose a sequence file. The default callback VI returns the path to the selected file in the Sequence Path control, returns `TRUE` in **Cancelled?** if the operator cancels the dialog box, and always returns `no error` in the error out control. If the Select Sequence callback VI returns `TRUE` in **Cancelled?**, the Test Executive cancels the operation and keeps the currently loaded sequence. If the callback VI returns an error in the error out control, the Test Executive cancels the operation and displays the error in the Test Display. To use a custom Open Sequence dialog box or to restrict access to sequences based upon the users privilege level, modify the default Select Sequence callback VI.

Open Sequence

The Test Executive calls the Open Sequence callback VI when it opens a new sequence. The Test Executive installs the default `Open Sequence Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the test sequence that was opened.
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.
Output	Sequence Info String	String	Contains a string that appears in the Sequence Info string indicator on the operator interface panel. This string contains a description of the sequence or instructions to the operator.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Open Sequence callback VI performs no action. If the operation is successful, the default callback VI returns the sequence description in the Sequence Info String control and returns `no error` in the error out control. If the Open Sequence callback VI returns an error or warning to the Test Executive, the error appears in the Test Display. The string returned in Sequence Info String appears in the Sequence Info box

on the operator interface panel. You can modify this VI to perform initialization functions or to log sequence file usage.

Close Sequence

The Test Executive calls the Close Sequence callback VI immediately before it closes a sequence. The Test Executive installs the default `Close Sequence Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the test sequence that is being closed.
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Close Sequence callback VI performs no action. If the close operation is successful, it returns `no error` in the error out control. If the callback VI returns an error to the Test Executive, the error appears in the Test Display. You modify this callback VI to perform cleanup functions or to log sequence file usage.

Save Sequence

The Test Executive calls the Save Sequence callback VI immediately after it saves a sequence. The Test Executive installs the default `Save Sequence Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the test sequence that was saved.
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Save Sequence callback VI performs no action. If the save operation is successful, it returns `no error` in the error out control. If the callback VI returns an error to the Test Executive, the error appears in the Test Display. You modify this callback VI to log sequence file modifications.

Sequence Report

The Test Executive calls the Sequence Report callback VI when you click the **Sequence Report** button on the operator interface panel. The Test Executive installs the default `Sequence Report Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the test sequence currently shown in the Sequence Display.

The default Sequence Report callback VI generates an ASCII, spreadsheet-style report of the contents of the currently loaded sequence and prompts you to save the report to disk. You modify this callback VI to customize the contents of the Sequence Report ASCII file to meet your needs.

Exit

The Test Executive calls the Exit callback VI when you quit the Test Executive. If there is an open test sequence, the Test Executive closes it before calling the Exit callback VI. The Test Executive installs the default `Exit Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.

You modify the Exit callback VI to work with the Login callback VI to log usage of the Test Executive.

Sequence Callbacks

The Test Executive includes the following sequence callback VIs:

- Pre-UUT Loop
- Pre-UUT
- Post-UUT
- Post-UUT Loop
- Pre-Step
- Post-Step
- Test Report
- Post Run-Loop Test
- Test Failure
- Open Test VI

Each time you launch the Test Executive, it loads the appropriate sequence callback VIs by first reading the entries in the system configuration file, `testexec.ini`. If the Test Executive cannot find the system configuration file, it prompts you to find it. When you quit the Test Executive, it unloads the sequence callback VIs.

Although the Test Executive contains a default set of sequence callback VIs, each sequence has its own custom set of sequence callback VIs. If a sequence uses a custom callback VI, the Test Executive loads the VI when it opens the sequence and unloads it when it closes the sequence. While the sequence is open, the custom callback VIs override the corresponding default callback VIs. You specify custom callback VIs for a sequence by selecting **Sequence Options...** in the Sequence Editor. For more information, see Chapter 4, *Creating Tests and Test Sequences*.

When you create custom callback VIs that individual sequences specify, do not use names that are the same as the default callback VIs specified in your `testexec.ini` file.

The Test Executive calls the first seven sequence callback VIs—Pre-UUT Loop, Pre-UUT, Post-UUT, Post-UUT Loop, Pre-Step, Post-Step, and Test Report—during the execution of a test sequence. Figure 5-1 shows how the Test Executive calls these callback VIs during a UUT Test loop.

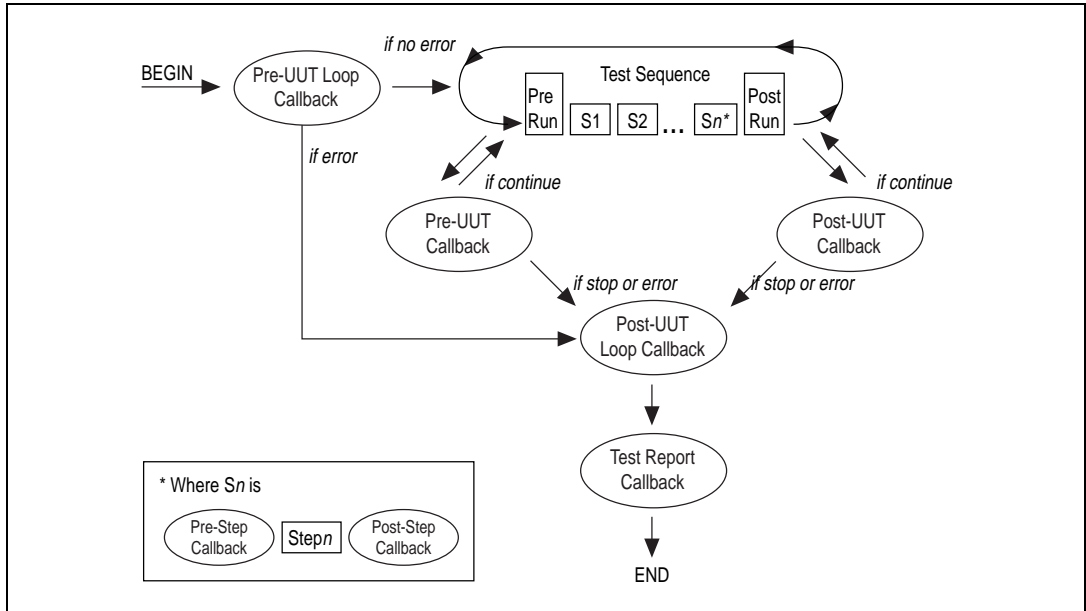


Figure 5-1. Flow of Sequence Callback VIs in a UUT Test Loop

The sequence callback VIs handle operations such as putting up a UUT Information dialog box at the start of a UUT Test, putting up a PASS/FAIL banner at the end of a UUT Test, and generating an ASCII Test Report at the end of a UUT Test Loop. By modifying these default sequence callback VIs or by specifying custom sequence callback VIs, you create customized UUT Information dialog boxes, customized PASS/FAIL banners, or customized Test Reports.

When a step with a FAIL Action of `Callback` fails, the Test Executive engine calls the Test Failure callback VI to determine what action it should take. After the user clicks the **Run Step(s)** or **Loop Step(s)** button to run or loop an individual step, the Test Executive engine calls the Post Run-Loop Test callback VI and passes the step results to it.

The Sequence Editor calls the last sequence callback VI, Open Test VI, when you click the **Edit Test VI** button or double-click a LabVIEW test in the sequence list. The default Open Test VI callback opens the front panel of the selected LabVIEW test, so you can edit it. You replace this default callback VI with one that automatically configures the call to the test VI or performs other LabVIEW test editing operations.

The rest of this section contains a detailed description of the calling interface for each sequence callback VI.



Note In the parameter tables that follow, the type of some inputs and outputs is listed as `TYPEDEF-xxx.ct1`. This value indicates that the input or output is a Test Executive typedef control. For a detailed examination of these typedef controls, see the [Test Executive Typedef Controls](#) section of this chapter.

Pre-UUT Loop

The Test Executive calls the Pre-UUT Loop callback VI before it tests the first UUT in a UUT Test Loop. You initiate a UUT Test Loop by clicking the **Test UUT** button. The Test Executive installs the default `Pre-UUT Loop Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the top-level test sequence in the hierarchy being executed.
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Pre-UUT Loop callback VI performs no action and returns `no error` in the error out control. If the Pre-UUT Loop callback VI returns an error to the Test Executive, it terminates the UUT Test Loop and displays the error in the Test Display. You modify this callback VI to perform appropriate initialization before a UUT Test Loop begins.

Pre-UUT

The Test Executive calls the Pre-UUT callback VI just before calling the `PreRun` VI at the beginning of each cycle of a UUT Test Loop. You initiate a UUT Test Loop by clicking the **Test UUT** button. The Test Executive installs the default `Pre-UUT Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	TYPEDEF-Sequence.ct1	Contains information about the top-level sequence in the hierarchy being executed.
Input	Previous UUT Info	String	Contains the UUT information for the last UUT that was tested. This input is empty for the first UUT in a UUT Test loop.
Input	Current Login Info	TYPEDEF-Login Info.ct1	Contains login information for the current user.
Input	UUT#	Numeric (I32)	Indicates how many UUTs have been tested in the current UUT Test Loop.
Output	Continue?	Boolean	Indicates whether to continue the UUT Test Loop.
Output	UUT Info	String	Contains user-supplied information about the next UUT to be tested.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Pre-UUT callback VI displays a UUT Info dialog box that prompts you to enter a serial number for the UUT about to be tested. If you click the **Stop** button on the default UUT Info dialog box, the callback VI sets the **Continue?** Boolean to FALSE and returns `no error` in the error out control. Otherwise, the callback VI copies the serial number into the UUT Info control, sets **Continue?** to TRUE, and returns `no error` in the error out control. The Test Executive ends the UUT Test Loop if the Pre-UUT callback VI returns FALSE in the **Continue?** control. If the callback VI returns an error to the Test Executive, it terminates the UUT Test Loop and displays the error in the Test Display.

Similar to defining custom login information in the Login callback VI, you also can define custom UUT information in the Pre-UUT callback VI. For more information, see the [Login](#) section earlier in this chapter.

Post-UUT

The Test Executive calls the Post-UUT callback VI just after calling the PostRun VI at the end of each cycle of a UUT Test Loop. You initiate a UUT Test Loop by clicking the **Test UUT** button. The Test Executive installs the default `Post-UUT Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	TYPEDEF-Sequence.ct1	Contains information about the top-level sequence in the hierarchy being executed.
Input	UUT Results	TYPEDEF-UUT Results.ct1	Contains user-supplied UUT information and raw UUT test results.
Input	Current Login Info	TYPEDEF-Login Info.ct1	Contains login information for the current user.
Input	UUT#	Numeric (I32)	Indicates how many UUTs have been tested in the current UUT Test Loop.
Output	Continue?	Boolean	Indicates whether to continue the UUT Test Loop.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

If UUT testing is successful, the default Post-UUT callback VI displays a PASS banner if all the steps in the sequence passed or a FAIL banner if any step in the sequence failed and its `Step Fail = Seq. Fail?` flag was set to TRUE. If you abort testing for the UUT, the default Post-UUT callback VI displays an ABORT banner. The default Post-UUT callback VI always returns TRUE in the **Continue?** control. If the Post-UUT callback VI returns FALSE in the **Continue?** control, the Test Executive ends the UUT Test Loop, just as it does when the Pre-UUT callback VI returns FALSE in **Continue?** If the Post-UUT callback VI returns an error in the error out control, the Test Executive terminates the UUT Test Loop and displays the error in the Test Display.

You can modify the Post-UUT callback VI to perform custom actions, such as logging result data on a per-UUT basis to a file or a database.

Post-UUT Loop

The Test Executive calls the Post-UUT Loop callback VI when you terminate a UUT Test Loop. You initiate a UUT Test Loop by clicking the **Test UUT** button. The Test Executive installs the default `Post-UUT Loop Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the current test sequence.
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Post-UUT Loop callback VI performs no action and returns `no error` in the error out control. If the callback VI returns an error to the Test Executive, it displays the error in the Test Display. You modify the Post-UUT Loop callback VI to perform appropriate cleanup after a UUT Test Loop ends.

Pre-Step and Post-Step Callbacks

Pre-Step and Post-Step are sequence callbacks that execute before and after each test step, respectively. The Test Executive installs the default `Pre-Step Callback.vi` and `Post-Step Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory.

The Pre-Step Callback uses the following calling interface.

Type	Name	Typedef	Description
Input	Execution Mode	<code>TYPEDEF-Execution Mode.ct1</code>	Specifies the execution mode under which the callback is being run.
Input	Current Login Info	<code>TYPEDEF-Login Info.ct1</code>	Contains login information for the current user.

Type	Name	Typedef	Description
Input	Invocation Info	TYPEDEF-Invocation Info.ct1	Contains the invocation information used for the associated test step.
Input	Runtime Status In	TYPEDEF-Runtime Status.ct1	Contains runtime information like resource path, function name (for C functions), and so on for the associated test step.
Output	Runtime Status Out	TYPEDEF-Runtime Status.ct1	Contains runtime information like resource path, function name (for C functions), and so on for the associated test step.
Output	error out	Standard LabVIEW error output cluster	Indicates whether an error or warning occurred in the callback VI.

The default Pre-Step Callback performs no action and returns `no error` in the error out control.

The Post-Step Callback uses the following calling interface.

Type	Name	Typedef	Description
Input	Execution Mode	TYPEDEF-Execution Mode.ct1	Specifies the execution mode under which the callback is being run.
Input	Current Login Info	TYPEDEF-Login Info.ct1	Contains login information for the current user.
Input	Invocation Info	TYPEDEF-Invocation Info.ct1	Contains the invocation information used for the associated test step
Input	Runtime Status In	TYPEDEF-Runtime Status.ct1	Contains runtime information like resource path, function name (for C functions), and so on for the associated test step.
Output	Test Result	TYPEDEF-Test Result.ct1	Contains the result information from the associated test step.

Type	Name	Typedef	Description
Output	Runtime Status Out	TYPEDEF- <code>Runtime Status.ct1</code>	Contains runtime information like resource path, function name (for C functions), and so on for the associated test step.
Output	error out	Standard LabVIEW error output cluster	Indicates whether an error or warning occurred in the callback VI.

The default Post-Step Callback performs no action and returns `no error` in the error out control.

Test Report

The Test Executive calls the Test Report callback VI after calling the Post-UUT Loop callback VI at the end of a UUT Test Loop. You initiate a UUT Test Loop by clicking the **Test UUT** button. The Test Executive also calls this callback VI at the end of a Single Pass Test, which you initiate by clicking the **Single Pass** button. The Test Executive installs the default `Test Report Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Array	Array of TYPEDEF- <code>Sequence.ct1</code>	Contains information about the currently loaded test sequences.
Input	Top-Level Sequence Index	Numeric (I32)	Specifies which sequence in the Sequence Array input is the top-level sequence in the hierarchy being executed.
Input	Current Login Info	TYPEDEF- <code>Login Info.ct1</code>	Contains login information for the current user.
Input	Single Pass?	Boolean	TRUE if called from Single Pass mode, FALSE if called from Test UUT mode.

Type	Name	Typedef	Description
Output	Test Report	String	Contains a report string detailing the test results for each UUT that was tested.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Test Report callback VI generates a formatted, spreadsheet-style report string and returns it in the Test Report control. If the sequence has a report file, the Test Report callback VI appends or overwrites the report file, depending on the Report File Mode. You view the report string on the operator interface panel by clicking the **View Test Report** button. You can modify the Test Report callback VI to generate the report string in a different format or to perform other actions, such as logging result data to a database.

The default Test Report Callback VI uses the following VIs to access the test results from a temporary file, which the Test Executive maintains. These VIs can be found in `CALLBACK.LLB` in the Test Executive installation directory:

- `Open UUT Results File.vi`
- `Read UUT Results File.vi`
- `Close UUT Results File.vi`

Post Run-Loop Test

The Test Executive calls the Post Run-Loop Test callback VI when you click the **Run Step(s)** or **Loop Step(s)** button on the main operator interface panel. The Test Executive installs the default `Post Run-Loop Test Callback.vi` in the `CALLBACK.LLB` VI Library and uses the following calling interface.

Type	Name	Typedef	Description
Input	Test #	Numeric (I32)	Contains the index of the run or looped test in the sequence.
Input	Sequence Info	<code>TYPEDEF-Sequence.ct1</code>	Contains information about the currently loaded test sequence.

Type	Name	Typedef	Description
Input	Current Login Info	TYPEDEF-Login Info.ct1	Contains login information for the current user.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Post Run-Loop Test callback VI does nothing and returns no error in the error out control. You can modify this callback VI to log or process the results of the step that was run or looped. You can use the same VIs used in the default Test Report Callback VI to access the results for the testing operation.

Test Failure

The Test Executive calls the Test Failure callback VI when a step with a FAIL Action of Callback fails. The Test Failure callback VI allows you to choose the failure action. The Test Executive installs the default Test Failure Callback.vi in the CALLBACK.LLB VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Failed Test Result	TYPEDEF-Test Result.ct1	Contains the test result for the failed test.
Input	Test #	Numeric (I32)	Contains the index of the failed test in the sequence.
Input	Sequence Info	TYPEDEF-Sequence.ct1	Contains information about the currently loaded test sequence.
Input	Current Login Info	TYPEDEF-Login Info.ct1	Contains login information for the current user.
Output	Action	Numeric (I32)	Indicates a continue, stop, or retry action.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Test Failure callback VI opens a dialog box that prompts you to choose one of three actions: Continue, Stop, or Retry. When you click a button on this dialog box, the default callback VI returns the selected action

in the Action control. The only values for Action are 0 (Continue), 1 (Stop), and 2 (Retry). When the Test Failure callback VI returns `Continue`, the Test Executive logs the step failure and continues to run the next test in the sequence. When the callback VI returns `Stop`, the Test Executive stops testing the current UUT. When the callback returns `Retry`, the Test Executive runs the failed step again. If the Test Failure callback returns an error in the error out control, the Test Executive stops running the test sequence and displays the error in the Test Display. The default Test Failure callback VI always returns `no error` in the error out control.

You can modify the default Test Failure callback VI if you want to use a custom dialog box, to handle the failure automatically, or to handle the failure differently depending on your privilege level.

Open Test VI

The Sequence Editor calls the Open Test VI callback VI when you click the **Edit Test VI** button in the Sequence Editor. The Test Executive installs the default `Open Test VI Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory and uses the following calling interface.

Type	Name	Typedef	Description
Input	Sequence Path	File Path	Contains the absolute path to the sequence file for the sequence that is being edited. If the sequence has not been saved, this input is empty.
Input	Test	<code>TYPEDEF-Sequence Element.ct1</code>	Contains information about the test that is being edited.
Output	Test Input	String	If filled, the Sequence Editor copies the string into the Input Buffer for the test that is being edited. If empty, the Sequence Editor ignores this output.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the callback VI.

The default Open Test VI callback opens the front panel of the LabVIEW test for the test that is being edited. This way, you can edit the LabVIEW

test from within the Sequence Editor. The default Open Test VI callback returns the empty string in the Test Input control and returns `no error` in the error out control. If the Open Test VI callback returns an error to the Test Executive, it displays the error in a message box.

You can modify the Open Test VI callback if you want to generate input data for the LabVIEW test that is being edited. For a demonstration of this technique, see the [Advanced Modifications](#) section of this chapter.

Test Executive Typedef Controls

The Test Executive install program installs a set of typedef controls in the LabVIEW `USER.LIB\` directory, allowing you to create callbacks and test VIs more easily. These typedef controls define some of the required callback VI inputs and outputs and all of the required and optional test VI inputs and outputs. The Test Executive typedef controls are available from the **User Controls** subpalette of the **Controls** palette after you install the Test Executive development version.

When creating or modifying a callback VI or LabVIEW test, remember that the Test Executive calls these VIs by name. Therefore, every required input or output on the called VI must match in name, type, and data direction with what the Test Executive expects. For example, if the Test Executive expects a callback VI to have a Boolean input control named “abc” on its front panel, the call to the VI fails if the name of the control is not “abc”, if the type is not Boolean, or if it is not a control. Control name comparisons are case sensitive, so “ABC” does not match “abc”. Also, make sure that the required control and indicator names do not have any extra spaces.

Typedefs for Callback VIs

To help make sure that your callback VI inputs and outputs have the right name, type, and data direction, use the following Test Executive typedef controls when you create or modify these VIs.

TYPEDEF - Login Info.ctl

`TYPEDEF-Login Info.ctl` contains user login information. The elements of this cluster are Operating level and User Info.

Operating level is an enum with the three privilege levels of the Test Executive: Developer, Technician, and Operator.

User Info is a string that contains information about the current Test Executive user. The default Login callback VI stores the name and password of the current operator in the User Info control.

TYPDEF - Sequence.ctl

`TYPDEF-Sequence.ctl` contains all the specifications for a sequence. The elements of this cluster are Elements, Sequence VIs, Stop on any failure, Enable Test Report Logging, Test report file, Report file mode, Description, Seq. load specification, and Path.

Elements is an array of `TYPDEF-Sequence Element.ctl` typedefs. This array contains the definitions for all the steps in the sequence. For more information, see the *TYPDEF - Sequence Element.ctl* section.

Sequence VIs is an array of file paths. This array contains the paths to the following sequence VIs in the following order.

index	VI
0	PreRun
1	PostRun
2	Pre-Step Callback VI
3	Post-Step Callback VI
4	Pre-UUT Loop Callback VI
5	Pre-UUT Callback VI
6	Post-UUT Loop Callback VI
7	Post-UUT Callback VI
8	Test Report Callback VI
9	Edit Test VI Callback VI
10	Post Run-Loop Test Callback VI
11	Test Failure Callback VI

Stop on any failure is a Boolean flag. If Stop on any failure is TRUE, the Test Executive stops the sequence execution whenever any test fails.

Enable Test Report Logging is a Boolean flag. If Enable Test Report Logging is TRUE, the default Test Report Callback VI logs the ASCII text report to the file specified in the Test report file.

Test report file is a file path that stores the path to the test report file.

Report file mode is a text ring that indicates how results should be stored to file, if Enable Test Report Logging is TRUE. You can set Report file mode to append or overwrite.

Description is a string containing a description of the test sequence.

Seq. load specification is a text ring that determines how the Test Executive loads the step resources for the sequence. This allows you to set Seq. load specification to Use each step's load spec., Pre-load all steps, or Dynamic-load all steps.

Path is a file path that stores the path to the sequence file. If the sequence has not been saved, Path is empty.

TYPEDEF - Sequence Element.ctl

TYPEDEF-Sequence Element.ctl contains all the specifications for a single step. The elements of this cluster are Element type, Name, VI path, Function name, Input buffer, Limit, Run mode, FAIL action, Loop count, Dependencies, Load specification, Input buffer?, Invocation info?, Show Panel?, Step Fail = Seq. Fail?, and Comment.

Element type is a text ring with four items: LabVIEW Test, C Test, GOTO, and Sequence. The setting of this ring indicates the type of the step.

Name is a string that specifies either the name of a step or the target for a GOTO step. No two steps in a sequence can have the same name. Before executing a sequence, the Test Executive parses the target of each GOTO step. If the Test Executive finds a GOTO target that does not match any step name in the sequence, it stops parsing and displays an error to the user.

VI path is a file path that specifies the resource for the step.

Function name is the name of the function found in the DLL or shared library identified in VI path to call for C tests.

Input buffer is a string. If Input buffer? is TRUE, the Test Executive passes the string data in Input buffer to the LabVIEW test or C test at run time.

Limit is a string that specifies a type of comparison. The Test Executive applies this comparison to the measurements returned by the LabVIEW Test or C Test to make a PASS/FAIL determination.

Run mode is a text ring with four items: Normal, Skip, Force PASS, and Force FAIL. If you set the Run mode of a step to Normal, the Test Executive runs the step's resource and applies the step's limit comparison to the results. If you set the Run mode of the step to Skip, Force PASS, or Force FAIL, the Test Executive does not run the step, and it sets the result for the step to SKIPPED, PASS, or FAIL, respectively.

FAIL action is a text ring with four items: Continue, Stop, Loop, and Callback. If you set the FAIL action of a step to Continue, the Test

Executive continues to the next step in the sequence when the step fails. Setting the FAIL action to `STOP` causes the Test Executive to stop testing the current UUT when the step fails. If you set the FAIL action to `LOOP`, the Test Executive enters a failure loop, which means that the Test Executive continues to execute the step until it passes or the maximum loop count for the step is reached. If you set the FAIL action to `CALLBACK`, the Test Executive calls the Test Failure callback VI to determine whether it should continue to the next step, stop testing the UUT, or run the failed step again.

Loop count is an integer that specifies the maximum number of times that the Test Executive should run the step inside a failure loop.

Dependencies is a string that specifies the dependency expression of a step. Before executing a sequence, the Test Executive parses the dependencies of each step in the sequence. If the Test Executive finds an invalid step name in any dependency expression, it stops parsing and displays an error to the operator.

Load specification is a text ring with two items: Pre-load and Dynamic-load. When the Test Executive opens a test sequence, it loads the resources for all steps with a Pre-load Load specification. The Test Executive does not load the resources for Dynamic-load steps until they are called during the execution of the test sequence.

Input buffer? is a Boolean flag. If Input buffer? is `TRUE`, the Test Executive passes the string data in Input buffer to the LabVIEW test or C test at run time. If Input buffer? is `FALSE`, the Test Executive does not pass Input buffer data to the LabVIEW test or C test.

Invocation info? is a Boolean flag. If Invocation info? is `TRUE`, the Test Executive passes invocation information to the LabVIEW test at run time. If Invocation info? is `FALSE`, the Test Executive does not pass invocation information to the LabVIEW test.

Show Panel? is a Boolean flag. If Show Panel? is `TRUE`, the Test Executive shows the panel of the LabVIEW test before running it and closes the panel afterward. If Show Panel? is `FALSE`, the Test Executive does not show the panel of the LabVIEW test while running it.

Step Fail = Seq. Fail? is a Boolean flag. If Step Fail = Seq. Fail? is `TRUE`, the sequence fails if the step fails. If Step Fail = Seq. Fail? is `FALSE`, the result of the step does not affect the result of the sequence.

Comment is a string containing a comment for the step.

TYPEDEF - UUT Results.ctl

`TYPEDEF-UUT Result.ctl` contains all the UUT ID information and sequence result information for a particular UUT. The elements of this cluster are UUT Info, UUT Abort, UUT Error, UUT Time, and Sequence Results.

UUT Info is a string that contains the UUT Information supplied by the user in the Pre-UUT Callback VI.

UUT Abort is a Boolean flag. If UUT Abort is TRUE, the user aborted testing on this UUT. If UUT Abort is FALSE, the user did not abort testing.

UUT Error is a Boolean flag. If UUT Error is TRUE, an error occurred during the testing of this UUT that caused the Test Executive to abort the UUT Test Loop. If UUT Error is FALSE, no error occurred during the testing of this UUT.

UUT Time is an integer that gives the total execution time for the UUT Test in milliseconds. The UUT Execution timer starts immediately before the PreRun VI starts and stops immediately after the PostRun VI stops. The timer resolution of your computer affects the accuracy of the timing information.

Sequence Results is an array of `TYPEDEF-Sequence Result.ctl` typedefs. This array contains the results for each sequence that was run during the testing process. The order of the results is the order in which the sequences were executed. The first element in the array is the result for the top-level sequence executed. For more information, see the following *TYPEDEF - Sequence Result.ctl* section.

TYPEDEF - Sequence Result.ctl

`TYPEDEF-Sequence Result.ctl` contains all the step result information for a particular sequence. The elements of this cluster are Sequence Path, PreRun Time, PostRun Time, Test Report Path, and Step Results.

Sequence Path contains the path to the sequence file.

PreRun Time is an integer that gives the execution time of the PreRun VI in milliseconds.

PostRun Time is an integer that gives the execution time for the PostRun VI in milliseconds.

Test Report Path contains the path to the test report file.

Step Results is an array of `TYPEDEF-Test Result.ctl` typedefs. This array contains the results for each step (excluding GOTO steps) that was run during the testing process. For more information, see the following *TYPEDEF - Test Result.ctl* section.

TYPEDEF - Test Result.ctl

`TYPEDEF-Test Result.ctl` contains all the specifications for a single step result. The elements of this cluster are Element Type, Name, Result, Step Resource, Comment, User Test Output, Execution Time, String Measurement, Numeric Measurement, radix, Low Limit, High Limit, Comparison, String Limit, Limit Specification, and Step Fail = Seq. Fail?.

Element Type is a text ring with four items: LabVIEW Test, C Test, GOTO, and Sequence. The setting of this ring indicates the type of the step.

Name is the step name of the step that produced the result.

Result is an enum with five items: PASS, FAIL, None, Skipped, and Unknown. PASS and FAIL mean that the step passed or failed, respectively. None means that the comparison type of the step was `Log only`, and therefore, no PASS/FAIL determination was made for the step. Skipped indicates that the step did not execute, and Unknown indicates that the Test Executive could not determine if the step passed or failed because the step has no limit specification.

Step Resource is a file path that specifies the resource for the step.

Comment is a string containing a comment generated by the LabVIEW test.

User Test Output is a string containing data generated by the LabVIEW test or C test.

Execution Time is an integer that gives the execution time for the step resource in milliseconds.

String Measurement is a string measurement that is transmitted by the LabVIEW test.

Numeric Measurement is a double-precision, floating-point measurement that is transmitted by the LabVIEW test or C test.

radix is a text ring with six values: fractional, scientific, decimal, hex, octal, and binary. The setting of radix indicates whether Numeric Measurement, Low Limit, and High Limit should appear in fractional, scientific, decimal, hexadecimal, octal, or binary notation, respectively.

Low Limit is a double-precision, floating-point number that the Test Executive compares with Numeric Measurement to make a PASS/FAIL determination for the step.

High Limit is a double-precision, floating-point number that the Test Executive compares with Low Limit and Numeric Measurement to make a PASS/FAIL determination for the step.

Comparison is an enum with 14 items: EQ (==), NE (!=), GT (>), LT (<), GE (>=), LE (<=), GTLT (> && <), GTLE (> && <=), GELT (>= && <), GELE (>= && <=), Boolean, String, Log only, and None. If Comparison is EQ, NE, GT, LT, GE, or LE, the Test Executive applies the indicated comparison to Numeric Measurement and Low Limit and sets Result accordingly. If Comparison is GTLT, GTLE, GELT, or GELE, the Test Executive applies the indicated range comparison to Numeric Measurement, Low Limit, and High Limit and sets Result accordingly. If Comparison is Boolean, the Test Executive sets Result to PASS or FAIL based on the Boolean result transmitted by the LabVIEW test or C test. If Comparison is String, the Test Executive sets Result to PASS if String Measurement equals String Limit and sets Result to FAIL otherwise. This string comparison is case sensitive. If Comparison is Log only, the Test Executive sets Result to None, logs Numeric Measurement and String Measurement, and applies no comparison. If Comparison is None, the Test Executive sets Result to None, logs no measurement, and applies no comparison.

String Limit is a string that the Test Executive compares with String Measurement to make a PASS/FAIL determination for the LabVIEW test.

Limit Specification is a string used to store the complete limit specification for the step.

Step Fail = Seq. Fail? is a Boolean flag. If Step Fail = Seq. Fail? is TRUE, the sequence fails if the step fails. If Step Fail = Seq. Fail? is FALSE, the result of the step does not affect the result of the sequence.

Typedefs for LabVIEW Tests

To help make sure that your LabVIEW test inputs and outputs have the right name, type, and data direction, use the following Test Executive typedef controls when you create or modify these VIs.

TYPEDEF - Invocation Info.ctl

`TYPEDEF-Invocation Info.ctl` contains run-time information that passes from the Test Executive to the LabVIEW test. The elements of this cluster are Test Name, Sequence Path, UUT Info, loop #, and UUT #.

Test Name is the name of the step that is currently running.

Sequence Path contains the absolute file path to the sequence file that is currently executing.

UUT Info is a string that contains the UUT Information supplied by the user in the Pre-UUT callback VI.

loop # is the number of times that this step has run within a failure loop. In a failure loop, the Test Executive repeatedly executes the step until it passes or the maximum loop count for the step is reached. loop # is 0 for the first run of the step, 1 for the second, and so on.

UUT # is an integer that identifies how many UUTs have been tested in this UUT Test Loop. UUT # is 0 for the first UUT, 1 for the second, and so on.

TYPEDEF - Input buffer.ctl

`TYPEDEF-Input buffer.ctl` consists of a string containing test-specific input data that passes from the Test Executive to the LabVIEW test at run time.

TYPEDEF - Mode.ctl

`TYPEDEF-Mode.ctl` consists of a test ring containing two items, run and config, that you use on the front panel of LabVIEW test shells. The Test Executive identifies the call mode of LabVIEW test shells and passes the appropriate input value, run or config, to `Mode.ctl`. For more information on LabVIEW test shells, see the [Advanced Modifications](#) section of this chapter.

TYPEDEF - Test Data.ctl

`TYPEDEF-Test Data.ctl` contains result information that is transmitted from the LabVIEW test to the Test Executive. The elements of this cluster are PASS/FAIL Flag, Numeric Measurement, String Measurement, User Output, and Comment.

PASS/FAIL Flag is set by the LabVIEW test to indicate whether the step passed or failed. When the limit specification for a step is Boolean, the Test Executive uses this element to make a PASS/FAIL determination.

Numeric Measurement is a double-precision, floating-point number. When the limit specification for a test is EQ (==), NE (!=), GT (>), LT (<), GE (>=), LE (<=), GTLT (> && <), GTLE (> && <=) GELT (>= && <), GELE (>= && <=), the Test Executive uses this element to make a PASS/FAIL determination. The value that the LabVIEW test passes out in Numeric Measurement is stored in the Test Result cluster of the step.

String Measurement is a string. When the limit specification for a step is String, the Test Executive uses this element to make a PASS/FAIL determination. The value that the LabVIEW test passes out in String Measurement is stored in the Test Result cluster of the test.

User Output is a string. The LabVIEW test stores data of any kind in this string by using the Flatten to String function. The data that the LabVIEW test passes out in User Output is stored in the Test Result cluster of the step.

Comment is a string. The value that the LabVIEW test passes out in Comment is stored in the Test Result cluster of the step. Use the Comment option for storing comments that you want to include in the test report.

Common Modifications

The following sections describe common modifications that you may want to make to the Test Executive. You can learn how to make these modifications by editing the default callback VIs in the CALLBACK.LLB VI library in the Test Executive installation directory. Before modifying any of the default callback VIs, make a backup copy of CALLBACK.LLB.

This section covers the following areas:

- Passwords
- PASS/FAIL/ABORT banners
- UUT Serial Number prompt
- Test report

Changing Passwords

You can change the passwords that determine the Test Executive operating level (Developer, Technician, or Operator) by modifying the default Login callback VI. (See the [Operating Levels](#) section in Chapter 1, [Introduction](#),

for a description of the operating modes.) To modify the default Login callback VI, open `Login Callback.vi` in `CALLBACK.LLB` in the Test Executive installation directory. Notice that `Login Callback.vi` has the required Login callback inputs and outputs on its front panel. This VI makes a subVI call to the Login VI. To examine the front panel of the Login VI, show the diagram of `Login Callback.vi`, pop up on the subVI call to Login, and select **Open Front Panel**.

To set the password that specifies the Developer level, examine the True case of the larger Case structure. Find the string constant, labeled `Developer Level Password`, which is wired into the Match Pattern function. Replace the string constant with the password you want to use to specify the Developer level. Subsequently, any password typed in that contains this sequence of characters sets the Test Executive to Developer level.

To set the password that specifies the Technician level, find the string constant labeled `Technician Level Password` inside the False case of the smaller Case structure. Replace this string constant with the password you want to use to specify Technician level. If the password entered at the prompt contains the character sequence in this string constant, the Test Executive sets the level to Technician. If a match is not found for either Developer or Technician passwords, the level defaults to Operator.



Note The password comparison is case sensitive.

Changing PASS/FAIL/ABORT Banners

You can change the PASS, FAIL, and/or ABORT Banner VIs, which display the result of a UUT Test, to show a custom screen. To do so, you must modify the default Post-UUT callback VI. To modify this VI, open `Post-UUT Callback.vi` in `CALLBACK.LLB` in the Test Executive installation directory. Notice that `Post-UUT Callback.vi` has the required Post-UUT callback VI inputs and outputs on its front panel. This VI makes subVI calls to PASS Banner, FAIL Banner, or ABORT Banner, depending on the UUT Test Results. On a color monitor, the PASS, FAIL, and ABORT banners have a colored background (green for pass, red for fail and abort), an **OK** button, and a free label in a large font containing the word PASS, FAIL, or ABORT.

You change the appearance of these banners by either revising the existing VIs (changing the message, colors, adding graphics, and so forth) or by replacing the PASS, FAIL, and/or ABORT Banner VIs with your own VIs. Notice that the execution palette and menus are hidden in the Banner VIs.

Changing the UUT Serial Number Prompt

You customize the prompt that asks for the UUT serial number by modifying the default Pre-UUT callback VI. To modify this VI, open `Pre-UUT Callback.vi` in `CALLBACK.LLB` in the Test Executive installation directory. Notice that `Pre-UUT Callback.vi` has the required Pre-UUT callback VI inputs and outputs on its front panel. This VI makes a subVI call to the UUT Information VI. To examine the front panel of the UUT Information VI, show the diagram of `Pre-UUT Callback.vi`, pop up on the subVI call to UUT Information, and select **Open Front Panel**. The front panel of this VI consists of a string control, its label (which prompts the user to Enter UUT Serial Number), and two buttons, **OK** and **STOP**.

To change the UUT serial number prompt message, retype the label on the front panel using the Labeling tool.

You can make other modifications to the serial number prompt, such as adding a routine that reads the serial number from a bar code reader via an RS-232 port, by editing the Pre-UUT callback VI.

Changing the Test Report

The Test Report is generated by the default Test Report callback VI and its subVIs. You can modify the test format of the Test Report to suit your needs.

To modify the default Test Report callback VI, open `Test Report Callback.vi` in `CALLBACK.LLB` in the Test Executive installation directory. Notice that `Test Report Callback.vi` has the required Test Report callback inputs and outputs on its front panel. This VI calls two subVIs, `Format Test Report` and `File Report`, and then institutes error checking procedures. The `File Report` VI sends the completed Test Report to a specified file and does not need to be changed to modify report format. Make any modifications in the `Format Test Report` VI block diagram or its main subVI, `Format UUT.vi`.

To change the Test Report header information, make the desired changes to the large `Concatenate Strings` function and its inputs in the `Format Test Report` VI. For example, you might want to include more information in the header. Stretch the `Concatenate Strings` function and add the desired string to an input. Another common change might be to add seconds to the time display. To do this, change the Boolean constant input to the `Get Date/Time String` VI from `FALSE` to `TRUE`.

The UUT Test results are generated in the For Loop of the Format Test Report VI. This VI converts the data read from a Test Executive temporary file to ASCII strings for inclusion in the Test Report.

Using Another Application for Report Generation

The format of a standard Test Report allows other applications to easily import the report. For example, because the Test Report uses tabs to delimit fields in a test result, each of these fields appears in a separate cell when you load the Test Report into a spreadsheet. Saving the Test Report to a file provides a simple mechanism for transferring the report to another application for further formatting of test results.

You may also want to use an interapplication communication mechanism to have the Test Executive automatically pass the Test Report to another application. You can use the communication VIs in LabVIEW for ActiveX Automation in Microsoft Windows, Apple Events on the Macintosh, or TCP/IP and UDP on all platforms. Using interapplication communication requires that you be familiar with the particular protocols and data formats the recipient application expects.

Advanced Modifications

This section describes advanced modifications that you can make to the Test Executive. The section covers result logging alternatives and using LabVIEW test shells.

Result Logging Alternatives

By default, the Test Executive logs the results for an entire UUT Test Loop to file. The default Test Report callback VI writes the Test Report string to the Report File as determined by the Report File Mode. This section discusses two result logging alternatives:

- Logging step results on a per-UUT basis
- Logging step results to a database using the SQL Tools (Windows only)

Logging Test Results on a Per-UUT Basis

You can modify the Test Executive to log UUT step results on a per-UUT basis as soon as a UUT completes testing. To set up this method, substitute a VI named `Per-UUT Logger Callback.vi` for your Post-UUT callback VI and `Test String Callback.vi` for your Test Report callback VI. Both

of these callback VIs are in the `CALLBACK.LLB` VI library in the Test Executive installation directory.

Per-UUT Logger Callback.vi

In addition to showing the PASS, FAIL, or ABORT banner, `Per-UUT Logger Callback.vi` creates a report string for the current UUT. If the current UUT is the first in the UUT Test Loop, `Per-UUT Logger Callback.vi` appends a report header to the report string and then, depending on the Report File Mode, either overwrites or appends this string to the report file. If the current UUT is not the first in the UUT Test Loop, `Per-UUT Logger Callback.vi` appends the report string to the Report File.

Test String Callback.vi

`Test String Callback.vi` creates the Test Report string from the UUT Test Loop results, but it does not write this Test Report string to the Report File.

Using the Edit Sequence Callbacks dialog box in the Sequence Editor allows you to make `Per-UUT Logger Callback.vi` and `Test String Callback.vi` the Post-UUT and Test Report callbacks for any sequence. Alternately, you can edit the Test Executive system configuration file, `testexec.ini`, and make these callback VIs the default Post-UUT and Test Report callbacks for all sequences. When you run a sequence that uses these two callback VIs, the Test Executive logs UUT Test results to file on a per-UUT basis.

Logging Results to a Database Using the LabVIEW SQL Tools (Windows only)

Using the LabVIEW SQL Tools, included in the Enterprise Connectivity Toolset, you can log your UUT, sequence, and step results to an SQL-compliant database. The `DATABASE` directory in the Test Executive installation directory contains an alternate `testexec.ini` file, as well as alternate operator interface and callback VIs that demonstrate the integration of the LabVIEW Test Executive and the SQL Tools. All source code is provided for these VIs, so you can modify them to meet your application-specific needs. These VIs were written using SQL Tools. If you want to use them with a previous version of the SQL Tools, edit them to use the equivalent SQL VIs from the previous version.

Modifications to the System Configuration File

To use the supplied database examples, you must add several preference values to the [Preferences] section of the `testexec.ini` file. A sample configuration file incorporating these modifications is located in the `DATABASE` directory in the Test Executive installation directory. The modifications are as follows:

- `EnableDatabaseSaving=TRUE` (or `FALSE`)
- `ConnectionString=a valid data source name for your system`
- `UUTTableName=name of table in which to store UUT results`
- `SequenceTableName=name of table in which to store sequence results`
- `StepTableName=name of table in which to store step results`

The Alternate Callback VIs

The `Create DB Tables Callback.vi` and `Per-UUT DB Logger Callback.vi` are alternate callback VIs that enable database logging on your Test Executive system. These VIs are found in the `DATABASE\CALLBACK.LLB` VI library in the Test Executive installation directory.

Create DB Tables Callback.vi

The `Create DB Tables Callback.vi` is a Pre-UUT Loop callback VI. It is responsible for creating three database tables in the database specified by the `ConnectionString` preference. These three tables are referred to as the UUT Results Table, the Sequence Results Table, and the Step Results Table. The names of the tables are specified by the preferences `UUTTableName`, `SequenceTableName`, and `StepTableName`, respectively. The `EnableDatabaseSaving` preference determines whether the tables are actually created.

The three tables contain the following fields:

- UUT Results Table
 - Serial Number
 - Result
 - Date
 - Operator
 - Top-level Sequence Name
 - Test Time

- Sequence Results Table
 - Serial Number
 - Sequence Name
 - Result
 - Pre-Run Execution Time
 - Post-Run Execution Time
- Step Results Table
 - Serial Number
 - Sequence Name
 - Step Name
 - Resource
 - Result
 - Execution Time
 - Comment
 - User Output
 - Comparison Type

If the tables already exist when the Create DB Tables Callback VI runs, the Test Executive uses the existing tables to record results.

Per-UUT DB Logger Callback.vi

The `Per-UUT DB Logger Callback.vi` is a Post-UUT callback VI. In addition to showing the PASS, FAIL, or ABORT banner, the VI logs the result information to the tables created by the `Create DB Tables Callback.vi`. Whether this VI logs the information to the database is determined by the `EnableDatabaseSaving` preference.

The `Per-UUT DB Logger Callback.vi` does not necessarily log all measurement information to the Step Results Table. If the step comparison is numeric, the Per-UUT DB Logger Callback VI logs numeric measurement and radix information. If the comparison is string, the Per-UUT DB Logger Callback VI logs string measurement and limit information. If the comparison is log only, the Per-UUT DB Logger Callback VI logs both the numeric and the string information.

Using the Sequence Options dialog box in the Sequence Editor, you can specify the Create DB Tables Callback VI and the Per-UUT DB Logger Callback VI as the Pre-UUT Loop and Post-UUT Callback VIs, respectively, for any sequence. Alternately, you can edit the Test Executive

system configuration file, `testexec.ini`, and make these callback VIs the default Pre-UUT Loop and Post-UUT callback VIs for all sequences. When you run a sequence that uses these two callback VIs (or any sequence if you modify the system configuration file), the Test Executive logs UUT Results, Sequence Results, and Step Results to the specified database on a per-UUT basis if `EnableDatabaseSaving` is `TRUE`.

Operator Interface VI

Use the alternate operator interface VI found in `DATABASE OPERATOR.LLB` in the Test Executive installation directory to browse the contents of the database tables specified in the system configuration file. The alternate operator interface VI has an additional button on its panel called **Database Browser....** Click this button to browse the database contents.

Using LabVIEW Test Shells

The Test Executive allows your end users to use LabVIEW Test shells to design test sequences using instrument drivers without doing any LabVIEW programming. A LabVIEW Test shell consists of a special kind of test VI that runs in two modes, config mode and run mode. A LabVIEW Test shell has the following calling interface.

Type	Name	Typedef	Description
Input	mode	<code>TYPEDEF-mode.ct1</code>	Determines the mode in which the LabVIEW Test shell runs. Notice that this control is not wired to the connector pane of the VI.
Input	Input buffer	<code>TYPEDEF-Input buffer.ct1</code>	Receives input data.
Output	Test Data	<code>TYPEDEF-Test Data.ct1</code>	Transmits data to its caller.
Output	error out	Standard LabVIEW error out cluster	Indicates whether an error or warning occurred in the Test VI shell.

Each LabVIEW Test shell must have these required inputs and outputs on its front panel and wired to the correct connector pane terminal, with the exception of the **mode** input. Like any other test VI, a LabVIEW Test shell also can contain additional controls and indicators on its front panel, as long as those controls and indicators are not wired to the connector pane of

the VI. Notice that the two required outputs for a LabVIEW Test shell, Test Data and error, are the same as the two required outputs for a test VI. Also notice that the Input buffer input, which is optional for test VIs, is required for a LabVIEW Test shell.

In run mode, the LabVIEW Test shell executes just like a test VI and transmits result and error information in Test Data and error. In run mode, the shell VI assumes that Input buffer contains a flattened cluster of input values and uses the Unflatten From String function to retrieve the input values.

In config mode, the LabVIEW Test shell acts like a dialog VI and allows the user to set the values of various input controls. In this mode, you click an **OK** button after setting the control values. Then, the LabVIEW Test shell bundles the control values into a cluster, flattens the cluster to a string, and returns the string in the User Output element of Test Data.

If you create a LabVIEW Test shell, use the Test Executive typedef controls. For more information on these controls, see the [Test Executive Typedef Controls](#) section earlier in this chapter.

Example Sequence Using LabVIEW Test Shells

When using LabVIEW Test shells to implement sequences, the sequence developer can configure instrument drivers without doing any LabVIEW programming. In a no-programming sequence, you must use an auto-configure Open Test VI callback and LabVIEW Test shells for every step. A sequence developer creates the sequence by using the Sequence Editor to add steps to the sequence, selecting LabVIEW Test shells for the steps, and configuring the LabVIEW Test shells by calling them in config mode. As an example of this process, perform the following operations.

1. Run the Test Executive and log in as a developer.
2. Click the **Edit** button to invoke the Sequence Editor.
3. Select **Sequence»Sequence Options...** to select an auto-configure Open Test VI callback for this sequence. In the Sequence VIs section, select the Edit Test VI callback from the callback ring. Click the **Browse** button and choose the auto-configure callback VI named `Configure Test VI Callback.vi` in the `CALLBACK.LLB` VI library in the Test Executive installation directory. Click the **OK** button to confirm your changes.
4. Back in the Sequence Editor, click the **New Step** button and enter `Fluke 45 Config` for the step name. Click the **Select Resource** button and select the LabVIEW Test shell named `Fluke 45 Config Shell`

VI in the TESTS\TEST_VIS.LLB VI library in the Test Executive installation directory.

5. To configure this LabVIEW Test shell, click the **Edit Test VI** button. This calls the auto-configure callback `Configure Test VI Callback VI`, which in turn calls the Test VI shell the `Fluke 45 Config Shell VI` in config mode. When this happens, the front panel of the `Fluke 45 Config Shell VI` appears and allows you to set the values of the four input controls on the panel.
6. When you have finished entering values, click the **OK** button. The `Fluke 45 Config Shell VI` flattens the values to a string and pass them back to the auto-configure callback VI. The auto-configure callback VI passes the same flattened string back to the Sequence Editor, which stores the string in the input buffer of the `Fluke 45 Config` step.
7. If you click the **Edit Test VI** button again, the process repeats, allowing you to examine or modify the input values you have entered.
8. Set the load specification for the step to Dynamic load.
9. Click the **OK** button on the Sequence Editor panel to confirm the changes you have made and return to the Test Executive.

The LabVIEW Test VI shell `Fluke 45 Config Shell` makes a subVI call to the test VI `Fluke 45 Config` in the TESTS\TEST_VIS.LLB VI library in the Test Executive installation directory. This subVI call has been configured to `Suspend` when called, so you can see the values that are passed to the test VI by the LabVIEW Test shell at run time.

10. Click the **Single Pass** button to run the test sequence once. The Test Executive executes the `Fluke 45 Config Shell VI` in run mode, passing it the flattened input string in the Input buffer. The `Fluke 45 Config Shell VI` unflattens the Input buffer string and passes the input values to the LabVIEW Test VI `Fluke 45 Config`. The `Fluke 45 Config VI` suspends when called, showing its front panel so you can examine the input values. Notice that the values are the same as those you entered when you configured the test in the Sequence Editor.
11. Click the **run** button to run the suspended test VI `Fluke 45 Config VI`. Click the **return to caller** button to terminate the test VI.
12. Save this example sequence under the name `NPROGRAM.SEQ`.

If you examine the auto-configure callback VI `Configure Test VI Callback`, you see that it calls the LabVIEW Test shell in config mode and passes it the current contents of Input buffer. This way, the shell VI displays the most recent user settings for the input controls. You can use this method

to create no-programming sequences of any length. Figure 5-2 shows the configuration and execution mechanisms for LabVIEW Test shells.

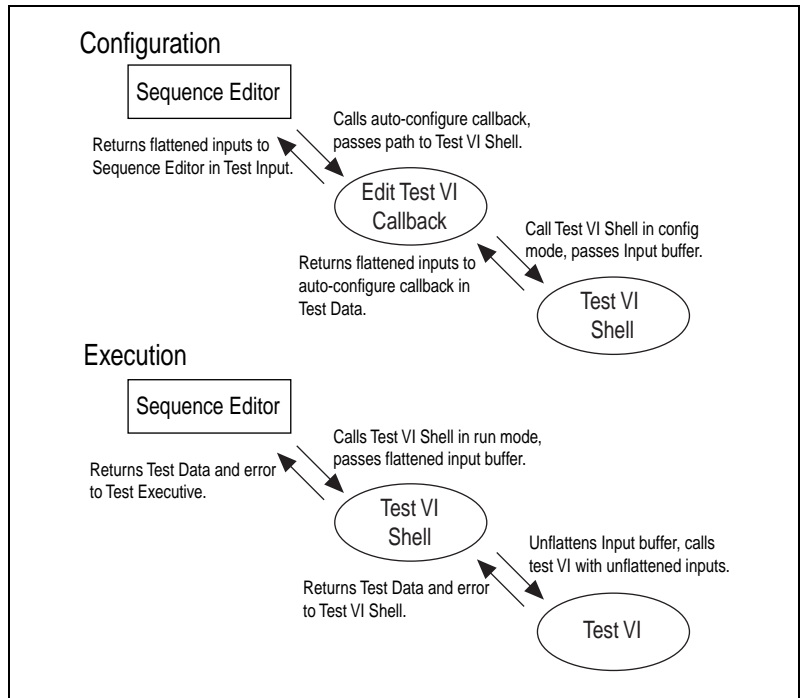


Figure 5-2. Test VI Shell Configuration and Execution

Deploying the Test Executive

This chapter explains how to build and deploy a LabVIEW Test Executive Run-Time System on a test station computer.

LabVIEW Test Executive Run-Time System

Using the LabVIEW Application Builder, you can build a Test Executive Run-Time System. This Run-Time System is an executable version of the Test Executive Operator Interface and Engine. You can distribute it to a test station computer without an accompanying LabVIEW installation, saving both money and installation space. When you purchase the Test Executive, you receive a license to create and use one copy of the Run-Time System. Please contact National Instruments for additional licenses.

Building a Run-Time System

Included with the Test Executive Development System installation is a LabVIEW Application Builder build script file. The setup program installs this script file in `lvexec.bld` in the Test Executive installation directory. To build a Run-Time System executable, load this script file into the LabVIEW Application Builder utility. Building this script produces an executable file named `lvtext.exe` (`lvtext` on Macintosh and Unix) in your Test Executive directory. This executable consists of the Text Executive Operator Interface and Engine built into an executable application.

The default build script file builds the Run-Time System using the Operator Interface `VI_OPERATOR.LLB\Test Executive` in the Test Executive installation directory. To build the Run-Time System using a different Operator Interface VI, modify the build script to specify a different top-level VI for the application. Refer to the *LabVIEW Application Builder Release Notes* for more information about using the Application Builder.



Note On Windows NT/98/95, you must install the LabVIEW Run-Time Engine to run the Test Executive Run-Time System and to run executables. Refer to the *LabVIEW Application Builder Release Notes* for more information on the LabVIEW Run-Time Engine.

Other Required Components for a Complete Run-Time System

The Run-Time System executable is not the only component required for a complete Test Executive installation. You also must provide the following customizable components:

- System Callback VIs
- Default Sequence Callback VIs
- `testexec.ini` file

Additionally, you also must provide any test sequences, resources such as test VIs and shared libraries, and non-default Sequence Callback VIs that you use with the Run-Time System installation. The following sections describe how to prepare these external components for distribution to a test station computer for use with a Run-Time System.

Callback and Test VIs

When preparing callback and test VIs for distribution to a test station computer for use with the Test Executive Run-Time System, you must save the VIs and their entire hierarchies to a new location. The new location can be either a directory or a VI library. You must then distribute the resulting directories or VI libraries to the test station computer.

To save a Test Executive VI for distribution with the Run-Time System, complete the following steps:

1. Select **Project»Test Executive»Utilities»Save Test Executive VI for Distribution...**
2. Using the first file dialog box, select the Test Executive VI you want to save for distribution. Click on the **OK** button. A second dialog box appears.
3. Using the second file dialog box, choose a new location for this VI and its subVIs. You also can create new directories or a VI library at this time.

4. Choose whether to save the VIs with or without their block diagrams. Saving without diagrams is not required, but it makes the resulting files smaller.
5. LabVIEW then saves your Test Executive VI and all of its subVIs (excluding those VIs already present in the Test Executive engine) to the directory or VI library you specified in step 3.
6. Use LabVIEW to mass compile the resulting VI library or directory.

You must follow the procedure listed in the previous paragraph for every VI required to create a complete Test Executive. These VIs include:

- All System Callback VIs and default Sequence Callback VIs specified in your `testexec.ini` file
- All Sequence Callback VIs and Pre-Run/Post-Run VIs needed by any sequence that you will execute with this Test Executive installation
- All test VIs needed by any sequence that you will execute with this Test Executive installation



Tip To streamline the saving process, create a VI that has all of your test VIs or callback VIs on its diagram. Then save this VI using the steps listed earlier in this section. Using this streamlined method, you do not have to follow the steps for saving VIs individually for every VI used with the Test Executive. Instead, you only perform the saving procedure once.

The testexec.ini File

The `testexec.ini` file tells the engine where to find all of the external components of the Test Executive. It must reside in the same directory as the Run-Time System executable file and provides the following information to the Test Executive engine:

- Locations of all system callback VIs
- Locations of all default sequence callback VIs
- Preference values

All path values must be valid for the test station computer and can be paths to local files or files on a shared network. For more information about the format of the `testexec.ini` file, refer to the [System Configuration File, testexec.ini](#) section of Chapter 5, [Modifying the Test Executive](#).

Test Sequences

Distribute the sequence files to be used with your Test Executive installation. Depending on the Sequence Path Specification used in each sequence file, you might need to update the files to contain the correct paths to test resources and non-default callback VIs, including Pre-Run and Post-Run VIs.

Shared Libraries (C Test Resources)

For shared library resources, distribute the DLL or shared library along with any run-time libraries required by that DLL or shared library.

Common Questions

This appendix includes a list of common questions you may have when using the Test Executive.

How easy is it to incorporate my existing LabVIEW tests into the Test Executive?

You must add two clusters to your existing LabVIEW VIs to pass status information to the Test Executive. Additionally, the VI must have the correct connector pane configuration required by the Test Executive. Chapter 4, [Creating Tests and Test Sequences](#), explains in detail how to add a LabVIEW test to the Test Executive.

How can I have individual steps displayed as they are executing?

In the Sequence Editor, select **Yes** in the **Show VI Panel at Run-Time** ring control for the step. For more information, refer to Chapter 4, [Creating Tests and Test Sequences](#).

Is it easy to modify the Test Executive to fit my own needs?

The Test Executive uses a modifiable operator interface VI and modifiable callback VIs to implement much of its functionality. To change the behavior of the Test Executive, you can edit or replace any of these VIs. For more information about modifying these VIs, refer to Chapter 5, [Modifying the Test Executive](#).

I have a test sequence that contains several step resources. How can I keep the Test Executive from loading all of the step resources into memory when I open the sequence? Is there a way to load a step resource into memory only when it is needed?

Normally, when the Test Executive opens a test sequence, it loads all the step resources that the sequence needs into memory. This process is called *pre-loading* the resources. If you do not want the Test Executive to pre-load a resource, use the Sequence Editor to set the load specification field for that step to `dynamic-load`. For more information, refer to Chapter 4, [Creating Tests and Test Sequences](#).

How can an operator or technician specify a filename to log test results?

Normally in the Test Executive, only the developer can modify the filename of the test report. This modification is typically made in the Sequence Editor. For the operator or technician to specify a file, you must first place a path control on the Test Executive front panel. Next, modify the block diagram of the operator interface VI to copy the path in this control to a global variable. Finally, modify the callback VI that logs test results (Post-UUT, Post-UUT Loop, or Test Report) to use the path in the global variable instead of the report file path. For more information about modifying the operator interface VI and the callback VIs, refer to Chapter 5, *Modifying the Test Executive*.

How can I build a filename from the serial number of the UUT?

The default Pre-UUT callback VI prompts the user for the serial number and passes it back to the Test Executive in the UUT Info output. The Test Executive stores this information in the UUT Results and passes it out to the Post-UUT, Post-UUT Loop, and Test Report callback VIs. You can modify any one of these callback VIs to retrieve the serial number, build the file path, and store the result data to file.

Does the Test Executive support the use of localized decimal points?

The Test Executive supports the use of localized decimal points for display purposes. If you set your LabVIEW Development System (or the run-time executable in the case of the Test Executive Run-Time System) preference to `Use localized decimal point`, the Test Executive displays and reports use your default system decimal point.



Note ASCII sequences files always denote numeric information using a period (“.”) as the decimal point.

Can I modify the Test Executive to have it open a particular sequence file when launched?

To have the Test Executive open a specific sequence file when launched, specify the path to the sequence file in the Path to Seq. File control on the Operator Interface VI. This control is located beneath the main controls on the front panel of the VI. After entering the full path to the sequence file, pop up on the control and select **Data Operations»Make Current Value Default**. Finally, save the changes to the VI.

When you specify a file path in the Path to Seq. File control, the Test Executive opens that sequence file when launched.

Can I call the Test Executive as a subVI?

You can call the Test Executive Operator Interface VI as a subVI. The VI has the standard error input and output clusters and a path input to specify the initial sequence file to load, if one is desired.

I am running the Test Executive under UNIX, and the text on the user interface panels is too large for the available space. Can I resize the text?

Depending on the configuration of your X server software, the text on the Test Executive user interface panels might be too large for the space allotted. If this is the case, change the default Application Font for LabVIEW (or the run-time executable in the case of the Test Executive Run-Time System) to a smaller size font. You can modify the font size in the LabVIEW Preferences dialog box. For more information about changing default fonts, see Chapter 7, *Customizing Your Environment*, in the *G Programming Reference Manual*.

Does the Test Executive reserve any particular error codes?

The Test Executive uses error codes 5000-5003. When creating LabVIEW Tests, C Tests, and callback VIs, do not use any of these error codes.

When National Instruments releases a new version of LabVIEW, will my Test Executive still work?

When you upgrade your LabVIEW installation, you must mass compile all of the Test Executive VI libraries and any test and callback VIs that you have created. Additionally, you must rebuild your Run-Time System executable using the new version of the LabVIEW Application Builder.

Can I customize the initial values of properties of new steps in the Sequence Editor?

To customize the initial values of properties of new steps in the Sequence Editor, edit the global variable `GLOBAL-New Step Default.vi` located in `ENGINE.LLB` in the Test Executive initialization directory. Modify the step properties in the global variable to the initial values you desire, select **Operate»Make Current Values Default**, and save the file. Any new steps you create in the Sequence Editor will then have the new initial property values.

Sequence Conversion Notes

This appendix describes the steps for converting a test sequence created with Version 4.0 or 5.0 of the Test Executive to the current version. If you have Version 3.0 sequences, convert them to Version 4.0 and then convert those sequences to the latest version. Please contact National Instruments technical support for assistance with this conversion.

Version 4.0 and 5.0 Conversion

Perform the following steps to convert sequence files from Version 4.0 or 5.0 to the latest version.

1. Use the Sequence File Converter to convert your sequence files to the new format.
2. Use LabVIEW to compile all Version 4.0 or 5.0 test VIs with the latest typedef controls.

Step 1—Use the 5.0 Sequence File Converter

The 5.0 Sequence File Converter is a LabVIEW VI that converts your Version 4.0 sequence files to Version 5.0.

To load the Sequence File Converter, perform the following steps.

1. Launch LabVIEW.
2. Select **Project»Test Executive»Utilities»Sequence File Converter...**

With the Sequence File Converter, you can open any 4.0 or 5.0 sequence file and convert it to the latest version. The following section explains the controls and indicators on the Sequence File Converter panel.

Controls

Open...

The **Open...** button displays a dialog box that prompts you to select a sequence file. You can select 4.0 or 5.0 sequence files from this dialog box.

Convert to 5.0...

When you click this button, the Sequence File Converter converts the currently loaded 4.0 or 5.0 sequence file to the latest version. It then displays a dialog box that prompts you to enter a name and location for the new 5.0 sequence file. When no 4.0 sequence file is loaded in the Sequence File Converter, this button is disabled.

Quit

The **Quit** button stops the Sequence File Converter operation.

Indicators

Sequence Name

The Sequence Name indicator displays the filename of the currently loaded sequence.

Sequence Version

The Sequence Version indicator displays the version of the currently loaded sequence.

Sequence File Path

The Sequence File Path indicator shows the complete file path of the currently loaded sequence.

Sequence Description

The Sequence Description indicator displays the sequence description for the currently loaded sequence. If no description has been entered for the sequence, this indicator will be empty.

Step 2—Compile Your Test VIs

To make sure that all your test VIs use the latest Test Executive typedef controls, you must compile those VIs with the LabVIEW after installing the Test Executive. To compile your test VIs, you must complete one of the following steps.

- Open each test VI in LabVIEW and save the changes to it.
- Use LabVIEW's mass compile operation (**File»Mass Compile...**) to compile an entire directory or library of VIs.



Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at www.natinst.com/support.

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from www.natinst.com/worldwide.

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Ontario) 905 785 0085,
Canada (Québec) 514 694 8521, China 0755 3904939,
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, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466,
Norway 32 27 73 00, Singapore 2265886, Spain (Madrid) 91 640 0085,
Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200,
United Kingdom 01635 523545

Glossary

A

ASCII American Standard Code for Information Interchange.

B

block diagram Pictorial description or representation of a program or algorithm. In LabVIEW, the block diagram, which consists of executable icons called nodes and wires that carry data between the nodes, is the source code for the VI. The block diagram resides in the Diagram window of the VI.

Boolean controls and indicators Front panel objects used to manipulate and display input and output Boolean (TRUE or FALSE) data. Several styles are available, such as switches, buttons, and LEDs.

C

callback VIs VIs designed for specific interface and data-logging operations. In the Test Executive, the system callback VIs handle interface operations, such as login and sequence opening and closing, while the sequence callback VIs handle run-time and edit-time events.

case One subdiagram of a Case structure.

Case structure Conditional branching control structure, which executes one and only one of its subdiagrams based on its input. It is the combination of the IF, THEN, ELSE, and CASE statements in control flow languages.

cluster A set of ordered, unindexed data elements of any data type including numeric, Boolean, string, array, or cluster. The elements must be all controls or all indicators.

cluster shell Front panel object that contains the elements of a cluster.

compile Process that converts high-level code to machine-executable code. LabVIEW automatically compiles VIs before they run for the first time after creation or alteration.

connector	Part of the VI or function node that contains its input and output terminals, through which data passes to and from the node.
control	Front panel object for entering data to a VI interactively or to a subVI programmatically.
Controls menu	Menu of controls and indicators.
current VI	VI whose Panel window, Diagram window, or Icon Editor window is the active window.

D

data logging	Generally, to acquire data and simultaneously store it in a disk file. LabVIEW file I/O functions can log data.
datalog file	File that stores data as a sequence of records of a single, arbitrary data type that you specify when you create the file. While all the records in a datalog file must be of a single type, that type can be complex. For instance, you can specify that each record is a cluster containing a string, a number, and an array.
dependency	Conditional execution of one step based on the result of another.
Diagram window	VI window that contains the block diagram code.
dialog box	An interactive screen with prompts in which you specify additional information needed to complete a command.

E

edit mode	The mode in which you create or edit a VI.
enumerations	Type controls and constants that are similar to ring controls and constants except in the way they display data. When an enumeration is wired to a Case structure, cases are named according to the enumeration's mnemonics rather than traditional numeric values.

F

- front panel The interactive user interface of a VI. Modeled from the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, and other controls and indicators.
- function Built-in execution element, comparable to an operator, function, or statement in a conventional language.

G

- Global variable A built-in LabVIEW object you use to easily access a given set of values throughout your LabVIEW application. A global variable is a special kind of VI with front panel controls that define the data type of the global variable.

I

- indicator Front panel object that displays output.

L

- label Text object used to name or describe other objects or regions on the front panel or block diagram.
- Labeling tool Tool used to create labels and enter text into text windows.
- LabVIEW Laboratory Virtual Instrument Engineering Workbench.
- Local variable A variable assigned to a front panel control or indicator on a VI. Once set up, a local variable always reads from or writes to the front panel control or indicator. You can use a local variable to read from an indicator or write to a control and in effect create an input-output control that LabVIEW does not normally have.

M

- MB Megabytes of memory.

N

numeric controls and indicators Front panel objects used to manipulate and display or input and output numeric data.

O

object Generic term for any item on the front panel or block diagram, including controls, nodes, wires, and imported pictures.

P

palette menu Menu that displays a palette of pictures that represent possible options.

Panel window VI window that contains the front panel, the execution palette and the icon/connector pane.

pop up To call up a special menu by clicking an object with the mouse. (In Windows, click the right mouse button to pop up.)

pull-down menus Menus accessed from a menu bar. Pull-down menu options are usually general in nature.

R

ring control Special numeric control that associates 32-bit integers, starting at 0 and increasing sequentially, with a series of text labels or graphics.

run mode The mode in which you execute a VI.

S

string controls and indicators Front panel objects used to manipulate and display or input and output text.

structure Program control element, such as a Sequence, Case, For Loop, or While Loop.

subdiagram	Block diagram within the border of a structure.
subVI	VI used in the block diagram of another VI; comparable to a subroutine.

T

terminal	Object or region on a node through which data passes.
tool	Special LabVIEW cursor you can use to perform specific operations.
top-level VI	VI at the top of the VI hierarchy. This term distinguishes the VI from its subVIs.
Type definition (typedef)	A master copy of a control. If you use a custom control, you can save it as a type definition and use that type definition in all your VIs. If you need to change that control, you can update the single type definition file instead of updating the control in every VI that uses it. A <i>strict</i> type definition can force everything about the control to be identical <i>everywhere</i> it is used, not just its data type.

U

UUT	unit under test
-----	-----------------

V

VI	Virtual instrument.
VI library	Special file that contains a collection of related VIs for a specific use.

W

While Loop	Post-iterative test loop structure that repeats a section of code until a condition is met. Comparable to a Do loop or a Repeat-Until loop in conventional programming languages.
wire	Data path between nodes.

Index

A

Abort banner. *See* PASS/FAIL/ABORT banners.

Abort button

default key assignment (table), 3-6

description, 3-3

Abort Loop button

default key assignment (table), 3-6

description, 3-3

adding steps, 2-6 to 2-7, 4-10. *See also* New Step button.

AND expressions, in dependencies, 4-23

architecture of Test Executive, 1-2 to 1-4

sequence callback VIs, 1-4

system callback VIs, 1-3 to 1-4

B

Bad Steps list box, 4-18

block diagram, operator interface VI, 5-4 to 5-5

C

C tests (Windows NT/98/95 and UNIX),

writing, 4-4 to 4-7

compiling test functions, 4-7

resizing text on user interface panel under UNIX, A-3

test data structure, 4-5 to 4-6

test error structure, 4-6 to 4-7

[Callback Paths] section of system configuration file

modifying, 5-1 to 5-3

patching, 5-2 to 5-3

callback VIs, 5-6 to 5-23. *See also* sequence callback VIs; system callback VIs.

modifying, 5-6

Post-Step and Pre-Step callback VIs, 5-17 to 5-19

requirements for Run-Time Systems, 6-2 to 6-3

Test Executive callback VI interface, 5-6

typedef controls, 5-23 to 5-29

Change Report File button, 4-18

Clear Step Status button

default key assignment (table), 3-6

description, 3-5

Clear Test Display button

default key assignment (table), 3-6

description, 3-5

Clear VI button, 4-18

Close button

default key assignment (table), 3-6

description, 3-1

Close Sequence callback VI, 5-10

command loop, operator interface VI, 5-5

comment field, editing, 4-16

COMMENT.SEQ example, 2-9

common questions about Test Executive, A-1 to A-3

Comparison Type ring control, 2-6

comparison type values

Limit Specification indicator (table), 4-12 to 4-13

Test Display (table), 3-9 to 3-10

compiling

recompiling Test Executive for new versions of LabVIEW, A-3

test functions, 4-7

test VIs, after converting from Version 4.0, B-2

comp_new.seq example, 2-9

computer_cvi.seq example, 2-9

COMPUTER.SEQ example, 2-9

- controls. *See also* indicators.
 - Abort button, 3-3
 - Abort Loop button, 3-3
 - Clear Step Status button, 3-5
 - Clear Test Display button, 3-5
 - Close button, 3-1
 - converting from Version 4.0 to
 - Version 5.0, B-1 to B-2
 - Edit button, 3-2
 - Function, 4-12
 - Input Buffer?, 4-14 to 4-15
 - Invocation Info?, 4-15
 - key assignments
 - operator interface (table), 3-6 to 3-7
 - Sequence Editor controls (table), 4-21
 - Login button, 3-2
 - Loop Step(s) button, 1-6, 2-3, 3-3 to 3-4
 - Max Loop Count, 4-14
 - Name, 4-11
 - New button, 3-2
 - Open button, 3-1, B-1 to B-2
 - Quit button, 2-4, 3-2, B-2
 - Report File Mode, 4-18
 - Run Mode, 3-4
 - Run Step(s) button, 1-6, 2-3, 3-3
 - Select VI control, 4-18
 - Sequence Path Specification, 4-17
 - Sequence Report button, 3-5
 - Sequence Runtime Updates?
 - checkbox, 3-4
 - Show Test VI Panel at Runtime?, 4-15
 - Single Pass button, 2-4, 3-2
 - Stop On Any Failure checkbox, 3-4
 - Test Runtime Updates? checkbox, 3-5
 - Test UUT button, 1-4, 2-2, 3-2
 - Type ring, 4-11
 - View Test Report button, 2-3, 3-5
 - conventions used in manual, *xiii*
 - converting test sequences, B-1 to B-2
 - compiling test VIs, B-2
 - Version 4.0 and 5.0 conversion, B-1 to B-2
 - Copy button, Dependency Editor, 4-25
 - Copy Steps button
 - description, 4-9
 - key assignment (table), 4-20
 - procedure for copying steps, 4-10
 - cpu_cvi.seq example, 2-9
 - cpu_diag.seq example, 2-9
 - cpu_lv.seq example, 2-9
 - Create DB Tables Callback.vi, 5-36 to 5-37
 - creating test sequences. *See* test sequences.
 - Cut button, Dependency Editor, 4-25
 - Cut Steps button
 - description, 4-9
 - key assignment (table), 4-20
- ## D
- decimal points, localized, A-2
 - Delete button, Dependency Editor, 4-23
 - Delete Steps button
 - description, 4-9
 - key assignment (table), 4-20
 - procedure for deleting steps, 4-10
 - Dependency Editor dialog box, 4-22 to 4-27
 - Cancel button, 4-25
 - complex dependencies, 4-24
 - copying, cutting, pasting, and undoing, 4-25
 - Dependency Editor key assignments (table), 4-26
 - example of setting dependencies, 2-7 to 2-8
 - OK button, 4-25
 - AND and OR expressions, 4-23 to 4-24
 - relationship among dependencies, run mode, and test flow, 4-26 to 4-27
 - rules for editing, 4-25

deploying Test Executive. *See* Run-Time System for Test Executive.
 Description button, 4-17
 Developer operating level, 1-7
 Development System for Test Executive, 2-2
 diagnostic resources, online, C-1
 documentation
 conventions used in manual, *xiii*
 related documentation, *xiv*

E

Edit button
 default key assignment (table), 3-6
 description, 3-2
 Edit Dependencies button. *See also* Dependency Editor dialog box.
 description, 4-15
 key assignment (table), 4-21
 setting dependencies, 2-7
 Edit menu, Sequence Editor, 4-20
 Edit Step Comment button
 description, 4-16
 key assignment (table), 4-21
 Edit Test VI button
 description, 4-15
 key assignment (table), 4-21
 editing test sequences. *See* Sequence Editor; test sequences.
 Enable Test Report Logging option, 4-17
 Error cluster, 4-3
 error codes reserved by Test Executive, A-3
 error messages, Test Display, 3-10
 error structure (test error structure), 4-6 to 4-7
 examining sample test program, 2-4 to 2-5
 execution model, 1-4 to 1-7
 modes, 1-4
 use of sequence callback VIs, 1-5 to 1-6
 Exit callback VI, 5-11
 exiting Test Executive. *See* Quit button.

F

FAIL Action
 description, 4-14
 key assignment (table), 4-21
 Fail banner. *See* PASS/FAIL/ABORT banners.
 File menu, Sequence Editor, 4-20
 front panel, operator interface VI, 5-4
 Function control
 description, 4-12
 key assignment (table), 4-21

G

GOTO Conditions button
 description, 4-16
 key assignment (table), 4-21
 GOTO Target control
 description, 4-16
 key assignment (table), 4-21

I

indicators. *See also* controls.
 converting from Version 4.0 to Version 5.0, B-2
 Limit Specification, 4-12 to 4-13
 Load Specification, 4-13
 required
 Error cluster, 4-3
 Test Data cluster, 4-2
 Resource, 4-11 to 4-12
 Sequence Display, 3-7 to 3-8
 Sequence Information, 3-12
 Sequence Name, 3-12
 Status, 3-12
 Test Display, 3-8 to 3-11
 comparison values and relative limits (table), 3-9 to 3-10
 error messages, 3-10

- results of each step, 3-9 to 3-10
 - Test Report, 3-10 to 3-11
- input buffer, for LabVIEW tests, 4-3 to 4-4
- Input Buffer? control
 - description, 4-14 to 4-15
 - key assignment (table), 4-21
- Insert control
 - above and below radio buttons, 4-9
 - adding new step, 2-6
 - key assignment (table), 4-20
- Invocation Info? control
 - description, 4-15
 - key assignment (table), 4-21
- Invocation Information cluster, 4-4

K

- key assignment
 - Dependency Editor (table), 4-23
 - operator interface (table), 3-6 to 3-7
 - Sequence Editor control keys (table), 4-20 to 4-21
 - Sequence Editor menu shortcuts (table), 4-22

L

- LabVIEW SQL Tools for result logging, 5-35 to 5-38
 - alternate callback VIs, 5-36 to 5-38
 - Create DB Tables Callback.vi, 5-36 to 5-37
 - Operator Interface VI, 5-38
 - Per-UUT DB Logger Callback.vi, 5-37 to 5-38
 - modifications to system configuration file, 5-36
- LabVIEW Test Executive. *See* Test Executive.
- LabVIEW Test Executive Run-Time System. *See* Run-Time System for Test Executive.

- LabVIEW test shells, 5-38 to 5-41
 - calling interface, 5-38
 - config mode, 5-39
 - configuration and execution (figure), 5-41
 - example sequences, 5-39 to 5-41
 - run mode, 5-39
- LabVIEW tests
 - incorporating existing tests into Test Executive, A-1
 - optional inputs, 4-3 to 4-4
 - input buffer, 4-3 to 4-4
 - Invocation Information cluster, 4-4
 - required indicators, 4-2 to 4-3
 - Error cluster, 4-3
 - Test Data cluster, 4-2
 - typedef controls, 5-29 to 5-31
 - writing, 4-1 to 4-4
- limit specification, configuring, 2-6. *See also* Set Limit Specification button.
- Limit Specification indicator, 4-12 to 4-13
- Load Specification indicator
 - description, 4-13
 - key assignment (table), 4-21
- localized decimal points, A-2
- logging. *See* result logging alternatives.
- Login button
 - default key assignment (table), 3-6
 - description, 3-2
- Login callback VI
 - calling interface, 5-7
 - customizing, 5-7
 - determining operating level, 2-2
 - overview, 1-3 to 1-4
- Login dialog box
 - default, 3-13
 - entering password, 2-1 to 2-2
- Loop Step(s) button, 2-3
 - default key assignment (table), 3-6
 - description, 3-3 to 3-4
 - typical use, 1-6
- Loop Step(s) mode, 1-4

M

manual. *See* documentation.
 mass editing of steps, 4-11
 Max Loop Count
 description, 4-14
 key assignment (table), 4-21
 modifying Test Executive. *See* Test Executive.
 multiple steps, running, 2-3

N

Name control
 description, 4-11
 key assignment (table), 4-21
 National Instruments Web support, C-1 to C-2
 New button
 default key assignment (table), 3-6
 description, 3-2
 New Step button
 description, 4-9
 editing test sequence, 2-5, 2-6
 key assignment (table), 4-20

O

online problem-solving and diagnostic
 resources, C-1
 Open button
 converting from Version 4.0 to Version
 5.0, B-1 to B-2
 default key assignment (table), 3-6
 description, 3-1
 Open Sequence callback VI
 calling interface, 5-9
 customizing, 5-9 to 5-10
 Open Test callback VI
 calling interface, 5-22
 customizing, 5-22 to 5-23
 typical test sequence, 1-6
 Open VI button, 4-18

operating levels
 changing to Technician level, 2-3
 determined by password entered in Login
 dialog box, 2-2
 Developer, 1-7
 Operator, 1-7
 Technician, 1-7
 operator dialog boxes, 3-13 to 3-15
 Login dialog box, 3-13
 Parsing Error Warning dialog box, 3-15
 PASS/FAIL/ABORT banners, 3-14
 Run-Time Error Warning dialog
 box, 3-15
 Select Sequence dialog box, 3-13
 Test Failed dialog box, 3-14
 UUT Information dialog box, 3-14
 [Operator Interface Path] section, system
 configuration file, 5-3
 operator interface VI
 block diagram, 5-4 to 5-5
 calling as subVI, A-3
 command loop, 5-5
 front panel, 5-4
 logging results with LabVIEW SQL
 Tools, 5-38
 modifying default VI, 5-4 to 5-5
 Operator level
 capabilities, 1-7
 changing to Technician level, 2-3
 OR expressions, in dependencies, 4-23

P

Parsing Error Warning dialog box, default,
 3-15
 PASS/FAIL/ABORT banners
 changing, 5-32
 default, 3-14
 passwords
 changing, 5-31 to 5-32
 determining operating level, 2-2

Paste button, Dependency Editor, 4-23

Paste Steps button
description, 4-9
key assignment (table), 4-20

Per-UUT DB Logger Callback.vi,
5-37 to 5-38

Per-UUT Logger Callback.vi, 5-34 to 5-35

Post-Run-Loop Test callback VI
calling interface, 5-20 to 5-21
customizing, 5-20 to 5-21

Post-run VIs
creating, 4-7
typical test sequence, 1-6

Post-Step callback VI
calling interface, 5-18 to 5-19
overview, 5-17

Post-UUT callback VI
calling interface, 5-16
customizing, 5-16 to 5-17
replacing with Per-UUT Logger
Callback.vi, 5-34 to 5-35
typical test sequence, 1-5

Post-UUT Loop callback VI
calling interface, 5-17
customizing, 5-17

Pre-run VIs
creating, 4-7
typical test sequence, 1-6

Pre-Step callback VI
calling interface, 5-17 to 5-18
overview, 5-17

Pre-UUT callback VI
calling interface, 5-15
customizing, 5-14 to 5-15
typical test sequence, 1-5

Pre-UUT Loop callback VI
calling interface, 5-15
customizing, 5-14 to 5-15

[Preferences] section, system configuration
file, 5-3

problem-solving and diagnostic resources,
online, C-1

Q

questions about Test Executive, A-1 to A-3

Quit button

converting from Version 4.0 to
Version 5.0, B-2
default key assignment (table), 3-6
description, 3-2
stopping execution of Test Executive, 2-4

R

repeating steps. *See* Loop Step(s) button.

Report File Mode control, 4-18

report generation. *See* Test Report.

Resource indicator

description, 4-11 to 4-12
key assignment (table), 4-21

result logging alternatives, 5-34 to 5-38

LabVIEW SQL Tools for logging to
database, 5-35 to 5-38

logging on Per-UUT basis, 5-34 to 5-35
specifying filename to log results, A-2

RTERROR.SEQ example, 2-9

Run Mode, Sequence Display, 3-7

Run Mode, Sequence Editor

key assignment (table), 4-21
options (table), 4-14
relationship among dependencies, Run
Mode, and test flow, 4-26 to 4-27

Run Mode control

default key assignment (table), 3-6
description, 3-4

Run Step(s) button

default key assignment (table), 3-6
description, 3-3
executing individual steps, 2-3
typical use, 1-6

Run Step(s) mode, 1-4
 Run-Time Error Warning dialog box,
 default, 3-15
 Run-Time System for Test Executive,
 6-1 to 6-4
 building, 6-1 to 6-2
 overview, 1-2, 6-1
 required components, 6-2 to 6-4
 callback and test VIs, 6-2 to 6-3
 shared libraries (C test resources),
 6-4
 test sequences, 6-4
 testexec.ini file, 6-3
 running test sequences. *See* test sequences.

S

Save Sequence callback VI
 calling interface, 5-10
 customizing, 5-11
 saving Test Executive VIs for distribution,
 6-2 to 6-3
 Select Resource button, 2-5
 Select Sequence callback VI
 calling interface, 5-8
 customizing, 5-9
 Select Sequence dialog box, default, 3-13
 Select VI control, 4-18
 sequence callback VIs
 flow of callback VIs in UUT test loop
 (*figure*), 5-13
 list of VIs, 1-4, 5-12
 modifying, 5-12 to 5-13
 Open Test
 calling interface, 5-22
 customizing, 5-22 to 5-23
 typical test sequence, 1-6
 Post-Run-Loop Test, 5-20 to 5-21
 Post-Step, 5-17 to 5-19

Post-UUT
 calling interface, 5-16
 customizing, 5-16 to 5-17
 replacing with Per-UUT Logger
 Callback.vi, 5-34 to 5-35
 typical test sequence, 1-5
 Post-UUT Loop, 5-17
 Pre-Step, 5-17 to 5-18
 Pre-UUT
 calling interface, 5-15
 customizing, 5-14 to 5-15
 typical test sequence, 1-5
 Pre-UUT Loop, 5-14 to 5-15
 Test Failure, 5-21 to 5-22
 Test Report
 calling interface, 5-19 to 5-20
 customizing, 5-19 to 5-20
 typical test sequence, 1-5 to 1-6
 test sequence and associated callback VIs
 (*figure*), 1-5
 typedefs, 5-23 to 5-29
 typical use in execution model, 1-5 to 1-7
 Sequence Description indicator,
 converting, B-2
 Sequence Display indicator, 3-7 to 3-8
 Run Mode field values (*table*), 3-7
 Step Status/Result field values (*table*), 3-8
 Sequence Editor. *See also* test sequences.
 control key assignments (*table*),
 4-20 to 4-21
 customizing initial values of properties of
 new steps, A-3
 Edit menu, 4-20
 editing dependencies, 4-22 to 4-27
 Cancel button, 4-25
 complex dependencies, 4-24
 copying, cutting, pasting, and
 undoing, 4-25
 Dependency Editor key assignments
 (*table*), 4-26
 OK button, 4-25

- AND and OR expressions,
 - 4-23 to 4-24
 - relationship among dependencies,
 - run mode, and test flow,
 - 4-26 to 4-27
 - rules for editing, 4-25
- File menu, 4-20
- invoking, 2-5
- mass editing, 4-11
- menu shortcuts (table), 4-22
- Sequence Errors, 4-18 to 4-20
- Sequence Options, 4-16 to 4-18
 - Change Report File button, 4-18
 - Description button, 4-17
 - Enable Test Report Logging, 4-17
 - Report File Mode control, 4-18
 - Sequence Load Specification, 4-16
 - Sequence Path Specification, 4-17
 - Sequence VIs, 4-18
 - Stop on any Failure checkbox, 4-17
- step editing elements, 4-9 to 4-11
 - Copy Steps button, 4-9
 - Cut Steps button, 4-9
 - Delete Steps button, 4-9
 - Insert, 4-9
 - New Step button, 4-9
 - Undo Step Edits button, 4-9
 - using editing elements, 4-10
- step editor controls, 4-11 to 4-16
 - Edit Dependencies button, 4-15
 - Edit Step Comment button, 4-16
 - Edit Test VI button, 4-15
 - FAIL Action, 4-14
 - Function, 4-11 to 4-12
 - GOTO Conditions button, 4-16
 - GOTO Target control, 4-16
 - Input Buffer? control, 4-14 to 4-15
 - Invocation Info? control, 4-15
 - Limit Specification indicator,
 - 4-12 to 4-13
 - Load Specification indicator, 4-13
 - Max Loop Count, 4-14
 - Name control, 4-11
 - Resource indicator, 4-11 to 4-12
 - Run Mode, 4-14
 - Show Test VI Panel at Runtime?
 - control, 4-15
 - Type ring control, 4-11
- Sequence Errors dialog box, 4-18 to 4-20
- sequence file, opening specific file while
 - launching, A-2
- Sequence File Path indicator, converting, B-2
- Sequence Information indicator, 3-12
- Sequence Load Specification control, 4-16
- Sequence Name indicator
 - converting from Version 4.0 to
 - Version 5.0, B-2
 - description, 3-12
- Sequence Options, 4-16 to 4-18
 - Change Report File button, 4-18
 - Description button, 4-17
 - Enable Test Report Logging, 4-17
 - Report File Mode control, 4-18
 - Sequence Load Specification, 4-16
 - Sequence Path Specification, 4-17
 - Sequence VIs, 4-18
 - Stop on any Failure checkbox, 4-17
- Sequence Path Specification control, 4-17
- Sequence Report button
 - default key assignment (table), 3-7
 - description, 3-5
- Sequence Report callback VI, 5-11
- Sequence Runtime Updates? checkbox
 - default key assignment (table), 3-6
 - description, 3-4
- Sequence Version indicator, converting, B-2
- Sequence VIs option, 4-18
- sequences. *See* test sequences.
- Set Limit Specification button
 - description, 2-6
 - key assignment (table), 4-21
- Set Limit Specification dialog box, 2-6 to 2-7

shared libraries, for Run-Time Systems, 6-4

Show Test VI Panel at Runtime? control
description, 4-15
key assignment (table), 4-21

Single Pass button
default key assignment (table), 3-6
description, 3-2
executing test sequence, 2-4

Single Pass mode
executing individual steps, 2-3 to 2-4
flow of execution (figure), 1-4
purpose and use, 1-4
typical test sequence, 1-6

software-related resources, C-2

SQL Tools. *See* LabVIEW SQL Tools for result logging.

starting Test Executive, 2-1 to 2-2

Status indicator, Test Display (table), 3-12

Step Problems list box, 4-19

step resources, loading into memory, A-1

Step Status/Result field values (table), 3-8

steps
definition, 4-8
displaying individual steps, A-1
executing individual steps, 2-3 to 2-4
mass editing, 4-11
specifications, 4-8

Stop On Any Failure checkbox
default key assignment (table), 3-6
description, 3-4
setting, 4-17

structures
test data structure, 4-5 to 4-6
test error structure, 4-6 to 4-7

system callback VIs
Close Sequence, 5-10
Exit, 5-11
Login
calling interface, 5-7
customizing, 5-7

determining operating level, 2-2
purpose and use, 1-3 to 1-4

Open Sequence, 5-9 to 5-10

overview, 1-3 to 1-4, 5-6 to 5-7

Save Sequence, 5-10 to 5-11

Select Sequence, 5-8 to 5-9

Sequence Report, 5-11

typedefs, 5-23 to 5-29

system configuration file (testexec.ini), 5-1 to 5-3
[Callback Paths] section, 5-1 to 5-3
modifying for result logging, 5-36
[Operator Interface Path] section, 5-3
[Preferences] section, 5-3
requirements for Run-Time Systems, 6-3

T

technical support resources, C-1 to C-2

Technician operating level
capabilities, 1-7
changing to, 2-3

Test Data cluster, 4-2

test data structure, 4-5 to 4-6
parameters (table), 4-5 to 4-6
structure definition, 4-5

Test Display, 3-8 to 3-11
comparison type values (table), 3-9 to 3-10
error messages, 3-10
result of each test, 3-9 to 3-10
Test Report, 3-10 to 3-11

test error structure, 4-6 to 4-7

Test Executive. *See also* Run-Time System for Test Executive.
architecture, 1-2 to 1-4
sequence callback VIs, 1-4
system callback VIs, 1-3 to 1-4
calling as subVI, A-3
common questions, A-1 to A-3
Development System, 1-2

- execution model, 1-4 to 1-7
- features, 1-1
- modifying
 - callback VIs, 5-6 to 5-23
 - operator interface VI, 5-4 to 5-5
 - PASS/FAIL/ABORT banners, 5-32
 - passwords, 5-31 to 5-32
 - result logging alternatives, 5-34 to 5-38
 - system configuration file (testexec.ini), 5-1 to 5-3
- Test Report, 5-33 to 5-34
- typedef controls, 5-23 to 5-31
- using another application for report generation, 5-34
- using LabVIEW Test shells, 5-38 to 5-41
- UUT serial number prompt, 5-33
- operating levels, 1-7
- overview, 1-1 to 1-2
- quitting, 2-4
- recompiling for new versions of LabVIEW, A-3
- Run-Time System, 1-2
- starting, 2-1 to 2-2
- versions, 1-2
- Test Executive VIs, saving for distribution, 6-2 to 6-3
- Test Failed dialog box, default, 3-13
- Test Failure callback VI
 - calling interface, 5-21
 - customizing, 5-21 to 5-22
- test program, examining, 2-4 to 2-5
- Test Report
 - changing, 5-33 to 5-34
 - enabling logging, 4-17
 - example (figure), 3-11
 - generating with other applications, 5-34
- Test Report callback VI
 - calling interface, 5-19 to 5-20
 - customizing, 5-19 to 5-20
 - typical test sequence, 1-5 to 1-6
- Test Runtime Updates? checkbox
 - default key assignment (table), 3-7
 - description, 3-5
- test sequences. *See also* Sequence Editor.
 - components, 4-7
 - conversion notes, B-1 to B-2
 - compiling test VIs, B-2
 - Version 4.0 and 5.0 conversion, B-1 to B-2
 - creating and editing, 2-5 to 2-8, 4-8 to 4-27
 - adding steps, 2-6 to 2-7, 4-10
 - configuring limit specification, 2-6
 - copying steps, 4-10
 - creating steps, 2-5 to 2-6
 - deleting steps, 4-10
 - inserting steps, 4-9
 - mass editing, 4-11
 - modifying steps, 4-10
 - running the sequence, 2-8
 - setting dependencies, 2-7 to 2-8
 - definition, 4-7
 - examining test program, 2-4 to 2-5
 - example sequences, 2-9
 - opening and running, 2-1 to 2-4
 - changing to Technician level, 2-3
 - exiting Test Executive, 2-4
 - individual steps, 2-3 to 2-4
 - multiple steps, 2-3
 - Single Pass mode, 2-3 to 2-4
 - starting Test Executive, 2-1 to 2-2
 - steps for running, 2-2 to 2-4
 - opening specific sequence file while launching, A-2
 - requirements for Run-Time Systems, 6-4

test shells. *See* LabVIEW test shells.

Test String Callback.vi, 5-35

Test UUT button

- default key assignment (table), 3-6
- description, 3-2
- executing test sequence, 1-4, 2-2

Test UUT mode

- purpose and use, 1-4
- typical test sequence, 1-5 to 1-6

testexec.ini. *See* system configuration file (testexec.ini).

tests, writing

- C tests (Windows NT/98/95 and UNIX), 4-4 to 4-7
 - compiling test functions, 4-7
 - test data structure, 4-5 to 4-6
 - test error structure, 4-6 to 4-7
- LabVIEW tests, 4-1 to 4-4
 - optional inputs, 4-3 to 4-4
 - required indicators, 4-2 to 4-3
- VI Wizard, 4-1

tTestData structure

- definition, 4-5
- parameters (table), 4-5 to 4-6

Type ring control

- description, 4-11
- key assignment (table), 4-20

typedef controls, 5-23 to 5-31

- callback VIs, 5-23 to 5-29
- LabVIEW tests, 5-29 to 5-31
- overview, 5-23

TYPEDDEF-Input buffer.ctl, 5-30

TYPEDDEF-Invocation Info.ctl, 5-30

TYPEDDEF-Login Info.ctl., 5-23 to 5-24

TYPEDDEF-Mode.ctl, 5-30

TYPEDDEF-Sequence Element.ctl, 5-25 to 5-26

TYPEDDEF-Sequence Results.ctl, 5-27 to 5-28

TYPEDDEF-Sequence.ctl, 5-24 to 5-25

TYPEDDEF-Test Data.ctl, 5-30 to 5-31

TYPEDDEF-Test Results.ctl, 5-28 to 5-29

TYPEDDEF-UUT Results.ctl, 5-27

U

UNDEFINE.SEQ example, 2-9

Undo button, Dependency Editor, 4-23

Undo Step Edits button

- description, 4-9
- key assignment (table), 4-20

UNIX tests. *See* C tests (Windows NT/98/95 and UNIX).

UUT Information dialog box, default, 3-14

UUT serial number

- building filename from serial number, A-2
- changing prompt, 5-33

UUT Test mode. *See* Test UUT mode.

V

VI Wizard, 4-1

View Test Report button

- default key assignment (table), 3-6
- description, 3-5

W

Web support from National Instruments, C-1 to C-2

- online problem-solving and diagnostic resources, C-1
- software-related resources, C-2

Windows NT/98/95 and UNIX tests. *See* C tests (Windows NT/98/95 and UNIX).

wizard, for writing tests, 4-1

Worldwide technical support, C-2

writing tests. *See* test sequences; tests, writing.