TestStand™

Using TestStand

NATIONAL INSTRUMENTS™

July 2003 Edition
Part Number 323436A-01

Worldwide Technical Support and Product Information

`ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599, Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949, Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838, Czech Republic 420 2 2423 5774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9131 0918, Mexico 001 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 1800 300 800, Norway 47 0 66 90 76 60, Poland 48 0 22 3390 150, Portugal 351 210 311 210, Russia 7 095 238 7139, Singapore 65 6226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227, Thailand 662 992 7519, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to `techpubs@ni.com`.

Important Information

Conventions

The following conventions are used in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
	This icon denotes a tip, which alerts you to advisory information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
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.

Contents

1 Introduction to TestStand

National Instruments TestStand is a flexible, open test management framework for building, customizing, and deploying a full-featured test management system. This chapter provides an overview of the major TestStand components.

If you are using TestStand for the first time, National Instruments recommends that you begin by completing the tutorials in this manual. While it is useful to make quick references to other manuals and the online help as you learn and use TestStand, be aware that the *TestStand Reference Manual* generally assumes familiarity with the concepts and tutorials in this manual.

In Chapter 2, *Loading and Running Sequences*, you learn about the windows, menus, commands, and dialog boxes in TestStand. The *TestStand Help*, which you can access through the **Help** menu of the TestStand Sequence Editor or from the *TestStand Bookshelf*, contains topics devoted to each of the windows and components in TestStand. Use these resources to find answers to any questions that arise as you use the *Using TestStand* manual. to launch the *TestStand Bookshelf*, select **Start»Program Files»National Instruments»TestStand 3.0»Online Help»TestStand Bookshelf**.

The tutorials in this manual begin with a general introduction to the TestStand Sequence Editor and continue through sections devoted to building sequences in TestStand. Because the steps of the tutorials build on elements from previous tutorials, National Instruments recommends that you follow the chapter outline.

TestStand Overview

The *TestStand System and Architecture Overview Card* offers a complete architectural diagram of TestStand, including descriptions of the various system components as well as diagrams illustrating sequence file execution flow, Execution object structure, and sequence file structure. Use this card to familiarize yourself with TestStand and as a reference while reviewing this manual.

Major Software Components of TestStand

This section provides an overview of the major software components of TestStand.

TestStand Engine

The TestStand Engine plays a central role in the TestStand architecture. The TestStand Engine is a set of DLLs that export an extensive ActiveX Automation Application Programming Interface (API) for creating, editing, running, and debugging sequences. The TestStand Sequence Editor and User Interface (UI) Controls use the TestStand API. You can access the TestStand API from any programming environment that supports access to ActiveX automation servers; thus, you can even call the TestStand API from a code module. Documentation for the TestStand API is available in the *TestStand Help*.

TestStand Sequence Editor

The TestStand Sequence Editor is an application program in which you create, modify, and debug sequences. The sequence editor gives you easy access to powerful TestStand features such as step types and process models. The sequence editor uses debugging tools that you are familiar with in application development environments (ADEs) such as LabVIEW, LabWindows™/CVI™, and Microsoft Visual Studio .NET. These include breakpoints, single-stepping, stepping into or over function calls, tracing, a variable display, and a Watch Expression pane.

In the TestStand Sequence Editor, you can start multiple concurrent executions, execute multiple instances of the same sequence, and execute different sequences at the same time.

TestStand Operator Interfaces

TestStand includes operator interfaces developed in a variety of ADEs. Each operator interface is a separate application program. These interfaces—developed in LabVIEW, LabWindows/CVI, Microsoft Visual Basic .NET, C#, and C++ (MFC)—are fully customizable and are provided in both source and executable formats.

The TestStand Operator Interfaces are fully customizable. Like the TestStand Sequence Editor, the operator interfaces allow you to start multiple concurrent executions, set breakpoints, and single-step. However, the operator interfaces do not allow you to modify sequences, and they do not display sequence variables, sequence parameters, step properties, and so on. Operator interfaces are intended for use with deployed test systems and are designed to protect the integrity of your test sequences.

Refer to the *TestStand System and Architecture Overview Card* and the *TestStand User Interface Controls Reference Poster* for more information about operator interfaces. Refer to Chapter 9, *Creating Custom Operator Interfaces*, in the *TestStand Reference Manual* for more information about customizing operator interfaces.

Module Adapters

Most steps in a TestStand sequence invoke code in another sequence or in a code module. When invoking code in a code module, TestStand must know the type of code module, how to call it, and how to pass parameters to it. The different types of code modules include LabVIEW VIs; C functions in source, object, or library modules created in LabWindows/CVI; C/C++ functions in DLLs; objects in .NET assemblies; objects in ActiveX automation servers; and subroutines in HTBasic. TestStand must also know the list of parameters required by the code module. TestStand uses a *module adapter* to obtain this knowledge.

TestStand includes the following module adapters:

- **LabVIEW Adapter**—Calls LabVIEW VIs with a variety of connector panes.
- **LabWindows/CVI Adapter**—Calls C functions with a variety of parameter types. The functions can be in object files, library files, or DLLs. They can also be in source files that are in the project you are currently using in LabWindows/CVI.
- **C/C++ DLL Adapter**—Calls functions or methods in a DLL with a variety of parameter types, including National Instruments Measurement Studio classes.
- **.NET Adapter**—Calls methods and accesses the properties of objects in a .NET assembly.
- **ActiveX/COM Adapter**—Calls methods and accesses the properties of objects in an ActiveX server.
- **HTBasic Adapter**—Calls HTBasic subroutines.
- **Sequence Adapter**—Calls other TestStand sequences with parameters.

The module adapters contain other important information in addition to the calling convention and parameter lists. If the module adapter is specific to an ADE, the adapter knows how to open the ADE, how to create source code for a new code module in the ADE, and how to display the source for an existing code module in the ADE.

Refer to Chapter 5, *Module Adapters*, in the *TestStand Reference Manual*, for more information about the module adapters included in TestStand.

Process Models

Testing a *Unit Under Test (UUT)* requires more than just executing a set of tests. The test management system must also perform a series of operations before and after the test sequence executes in order to handle common test system tasks such as identifying the UUT, notifying the operator of pass/fail status, generating a test report, and logging results. These operations define the testing process. The set of such operations and their flow of execution is called a *process model*.

Process models are essential for writing different test sequences without repeating standard testing operations in each sequence. Modifications to the process model allow you to vary the testing process to suit the unique needs of your production line, your production site, or the systems and practices of your company.

TestStand provides a mechanism for defining a process model in the form of a *sequence file*. You can edit a process model just as you edit your other sequence files. TestStand ships with three fully functional process models: the Sequential model, the Batch model, and the Parallel model. You can use the Sequential model to run a test sequence on one UUT at a time. The Parallel and Batch models allow you to run the same test sequence on multiple UUTs at the same time.

A process model also defines a set of Execution entry points. Each entry point is a sequence in the process model file. Multiple entry points in a process model give you different options for invoking test sequences.

For example, the default TestStand process model, the Sequential model, provides two Execution entry points for invoking test sequences: Test UUTs and Single Pass. The Test UUTs entry point initiates a loop that repeatedly identifies and tests UUTs. The Single Pass entry point tests a single UUT without identifying it. Both entry points allow you to log results and produce reports.

For more information about process models, refer to Appendix A, *Process Model Architecture*, in the *TestStand Reference Manual*.

In the next chapter, you will begin the TestStand tutorials. The first tutorial demonstrates how to load and view a sequence file.

2

Loading and Running Sequences

In this chapter, you will learn about the different windows in the TestStand Sequence Editor and how to load and run sequences in TestStand.

Starting TestStand

Complete the following steps to start the TestStand Sequence Editor:

1. Launch the TestStand Sequence Editor by selecting **Start»Programs»National Instruments»TestStand 3.0»Sequence Editor**. After the sequence editor displays the main window, it launches the Login dialog box, as shown in Figure 2-1.

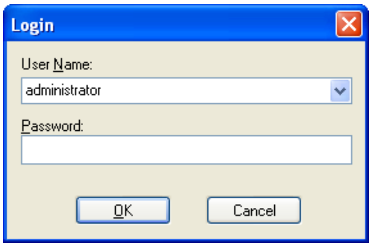
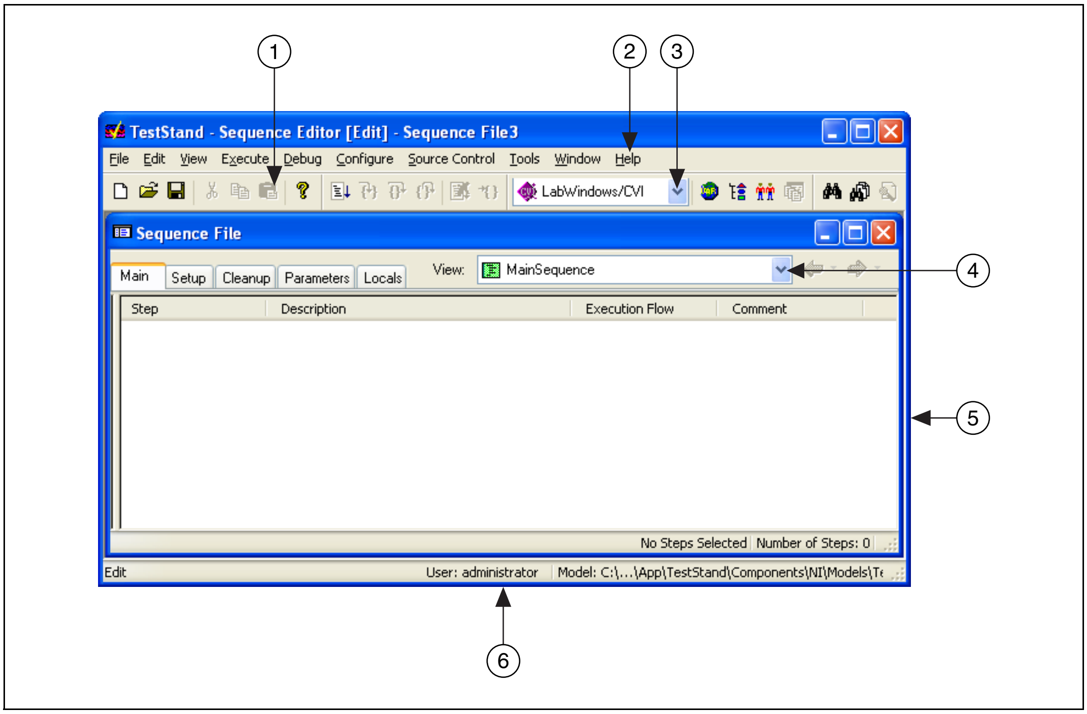


Figure 2-1. Login Dialog Box

2. Select the default user name, `administrator`, from the User Name ring control. In this default case, leave the **Password** field empty.
3. Click **OK** to begin. You should now see the Sequence Editor window, as shown in Figure 2-2.


1
2
3
4
5
6
TestStand - Sequence Editor [Edit] - Sequence File3
File Edit View Execute Debug Configure Source Control Tools Window Help
LabWindows/CVI
Sequence File
Main Setup Cleanup Parameters Locals
View: MainSequence
Step Description Execution Flow Comment
No Steps Selected Number of Steps: 0
Edit
User: administrator
Model: C:\...\App\TestStand\Components\NI\Models\Te

1	Toolbar	3	Adapter Ring Control	5	Development Workspace
2	Menu Bar	4	View Ring Control	6	Status Bar

Figure 2-2. Sequence Editor Main Window

Introduction to the Sequence Editor

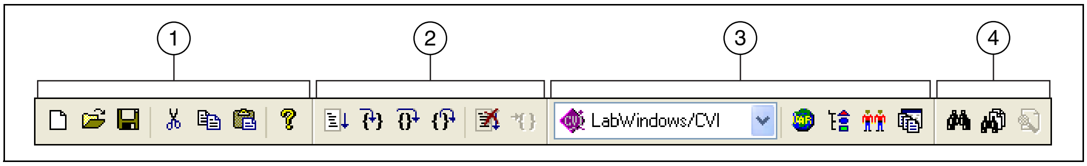
The Sequence Editor window has four main parts: the menu bar, the toolbar, the development workspace, and the status bar. You can learn more about the different sections of the Sequence Editor window in the *TestStand Help*.

Menu Bar

The menu bar contains the following menus: File, Edit, View, Execute, Debug, Configure, Source Control, Tools, Window, and Help. Browse the menus in the sequence editor to familiarize yourself with their contents. The *TestStand Help* contains a detailed explanation of each menu item.

Toolbar

The toolbar contains shortcuts to commonly used selections of the menu bar. As shown in Figure 2-3, the toolbar contains four sections: Standard, Debug, Environment, and Find.



1 Standard Section	2 Debug Section	3 Environment Section	4 Find Section

Figure 2-3. Sequence Editor Toolbar

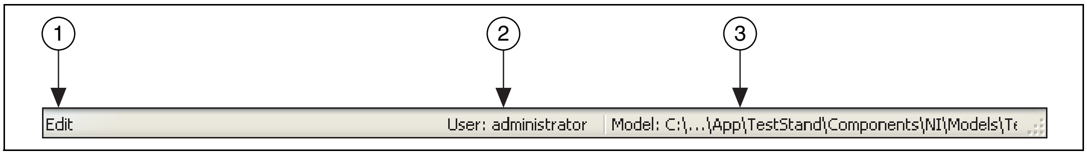
- **Standard Section**—Contains buttons for creating, loading, saving, cutting, and pasting sequence files.
- **Debug Section**—Contains buttons for executing a sequence, stepping into, stepping over, stepping out, pausing, and terminating an execution.
- **Environment Section**—Contains the Adapter ring control, buttons for opening other TestStand tools, and a button to bring an open workspace to the front.
- **Find Section**—Contains buttons for performing search and replace operations.

Development Workspace

The development workspace is the main area of the sequence editor. It is in this area that the sequence editor displays its windows.

Status Bar

The status bar displays common information in the sequence editor. As shown in Figure 2-4, the status bar contains three sections: Selection Help, User Name Display, and Model Display.



1 Selection Help	2 User Name Display	3 Model Display

Figure 2-4. Sequence Editor Status Bar

- **Selection Help**—Displays information about the currently selected menu item.
- **User Name Display**—Displays the user name of the current user.
- **Model Display**—Shows the pathname of the current process model file. You can open the current process model by double-clicking in this area.

About Sequences and Sequence Files

A sequence is made up of a series of steps. A *step* can initialize an instrument, perform a complex test, or change the flow of an execution. Steps can also jump to another step, call another sequence, call an external code module, or change the value of a *variable* or *property*.

The view for an individual sequence has five tabs: Main, Setup, Cleanup, Parameters, and Locals. Click each tab to view its contents. The Main, Setup, and Cleanup tabs each display a step group in the sequence. The steps in each step group help you perform the following tasks:

- **Main**—Test your UUTs.
- **Setup**—Initialize or configure your instruments, fixtures, and UUTs.
- **Cleanup**—Power down or release handles to your instruments, fixtures, and UUTs.

A sequence can also have parameters and any number of *local variables*. Parameters allow you to pass and receive values from the sequence. You can define parameters in the Parameters tab of the Sequence File window. Use local variables to hold values, maintain counts, hold intermediate values, and perform any other type of local data storage. Define local variables in the Locals tab of the Sequence File window.

A sequence file can contain any number of sequences along with sequence file global variables, which you can use to share data between multiple sequences within a sequence file.

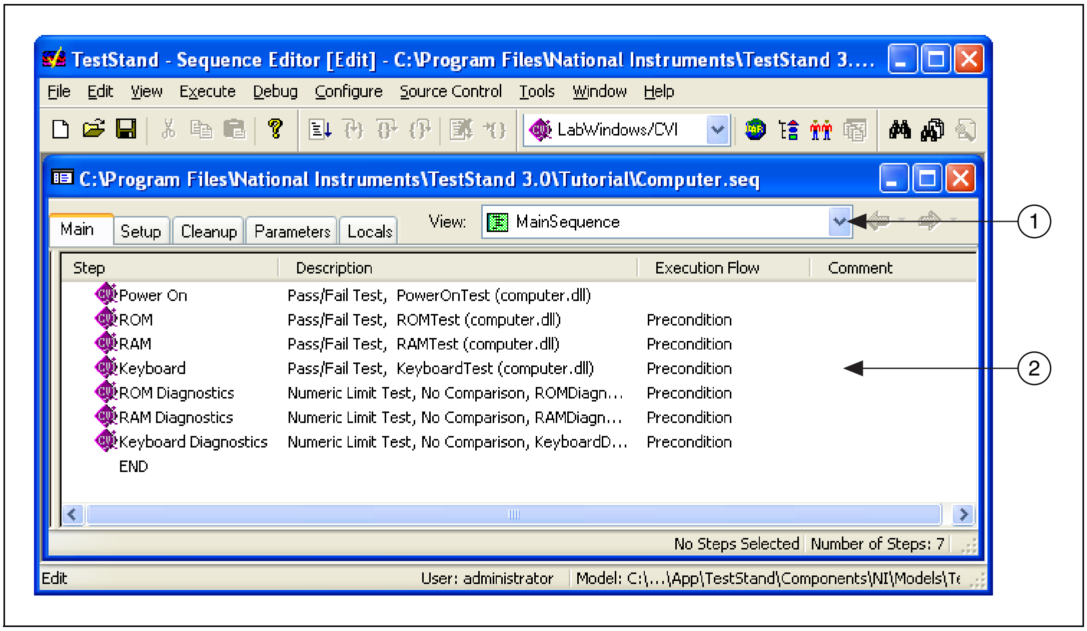
Loading a Sequence File

In this tutorial, you will load and view a sequence file.

1. Select **File»Open** to launch the Open dialog box, and then navigate to the `<TestStand>\Tutorial` subdirectory.
2. Select `Computer.seq` from the `Tutorial` subdirectory and click **Open**.

   The `Computer.seq` sequence file is a simulated test of your computer in which you can designate various hardware components to "fail" the test. The sequence runs tests that are functions in a dynamic link library (`.dll`) written with LabWindows/CVI.

   The sequence file opens in a separate window within the sequence editor. This window is called a *Sequence File window*, and is shown in Figure 2-5. You can load multiple sequence files into the sequence editor, which displays each file in its own Sequence File window.


TestStand - Sequence Editor [Edit] - C:\Program Files\National Instruments\TestStand 3....
File Edit View Execute Debug Configure Source Control Tools Window Help
LabWindows/CVI
C:\Program Files\National Instruments\TestStand 3.0\Tutorial\Computer.seq
Main Setup Cleanup Parameters Locals
View: MainSequence
1
2

Step	Description	Execution Flow	Comment
Power On	Pass/Fail Test, PowerOnTest (computer.dll)		
ROM	Pass/Fail Test, ROMTest (computer.dll)	Precondition	
RAM	Pass/Fail Test, RAMTest (computer.dll)	Precondition	
Keyboard	Pass/Fail Test, KeyboardTest (computer.dll)	Precondition	
ROM Diagnostics	Numeric Limit Test, No Comparison, ROMDiagn...	Precondition	
RAM Diagnostics	Numeric Limit Test, No Comparison, RAMDiagn...	Precondition	
Keyboard Diagnostics	Numeric Limit Test, No Comparison, KeyboardD...	Precondition	
END			

No Steps Selected | Number of Steps: 7

Edit | User: administrator | Model: C:\...\App\TestStand\Components\NI\Models\Te

1 View Ring Control	2 Sequence File Window

Figure 2-5. Sequence File Window for Computer.seq

3. Use the **View** ring control at the top right of the Sequence File window to select `MainSequence`. Figure 2-5 illustrates the View ring control.

   You can use the View ring control to view an individual sequence, a list of all sequences in the file, the sequence file global variables in the file, or the data and step types that you use in the file.

4. Browse the contents of each tab in the Sequence File window. Return to the Main tab when you finish.

Running a Sequence

When you run a sequence in the sequence editor, you are initiating an *execution*. In this section, you will examine two methods of running a sequence: running a sequence directly and running a sequence using a process model. The process model sequence contains a series of steps that specify the high-level flow of a sequence execution.

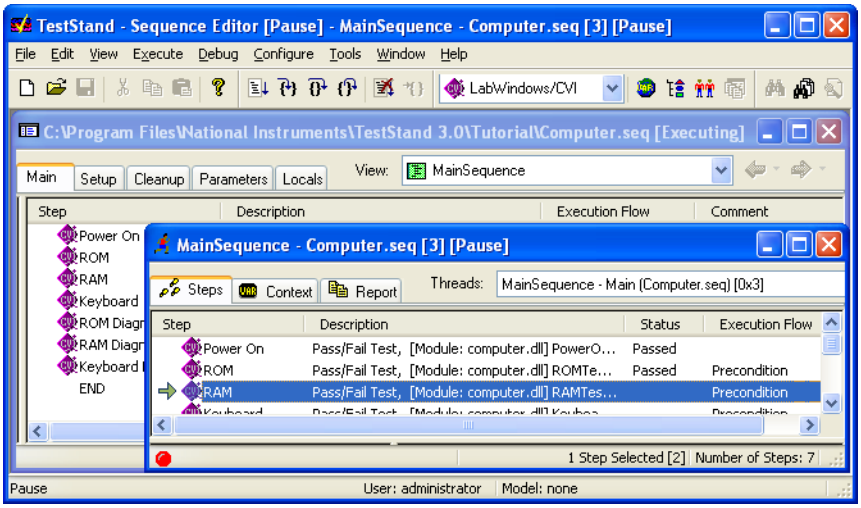
When you initiate an execution, the sequence editor opens an *Execution window*. Use the Execution window to view, or trace, steps as they execute, monitor the values of variables and properties, and examine the test report when the execution completes.

In trace mode, the Execution window displays the steps in the currently executing sequence. When the execution is suspended, the Execution window displays the next step to execute and provides debugging options. Trace mode is automatically enabled when you install TestStand.

Running a Sequence Directly

Complete the following steps to run `MainSequence` in the `Computer.seq` sequence file:

1. Select **Execute»Run MainSequence**. The sequence editor launches the Execution window, as shown in Figure 2-6.


TestStand - Sequence Editor [Pause] - MainSequence - Computer.seq [3] [Pause]
File
Edit
View
Execute
Debug
Configure
Tools
Window
Help
LabWindows/CVI
C:\Program Files\National Instruments\TestStand 3.0\Tutorial\Computer.seq [Executing]
Main
Setup
Cleanup
Parameters
Locals
View:
MainSequence
MainSequence - Computer.seq [3] [Pause]
Steps
Context
Report
Threads:
MainSequence - Main (Computer.seq) [0x3]
1 Step Selected [2]
Number of Steps: 7
Pause
User: administrator
Model: none

Step	Description	Execution Flow	Comment
Power On			
ROM			
RAM			
Keyboard			
ROM Diagr			
RAM Diagr			
Keyboard			
END			

Step	Description	Status	Execution Flow
Power On	Pass/Fail Test, [Module: computer.dll] PowerO…	Passed	
ROM	Pass/Fail Test, [Module: computer.dll] ROMTe…	Passed	Precondition
RAM	Pass/Fail Test, [Module: computer.dll] RAMTes…		Precondition

Figure 2-6. MainSequence Execution Window

After the execution starts, a Test Simulator dialog box opens in front of the Execution window. Figure 2-7 illustrates the Test Simulator dialog box for this tutorial.

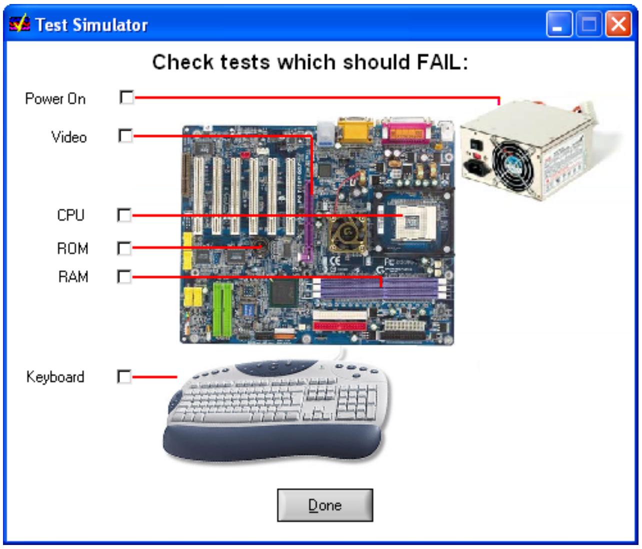


Figure 2-7. Test Simulator Dialog Box

This dialog box prompts you to designate the computer component(s), if any, that you want to return a value of `Fail` during the execution.

2. Select the RAM test by clicking its checkbox.
3. Click **Done**.
4. Observe the Execution window as it traces through the steps that TestStand runs.

   The sequence editor displays the progress of an execution by placing a yellow pointer icon to the left of the currently executing step in the Steps tab. The pointer icon is called the *execution pointer*. Notice that the status column for the RAM test contains the value `Failed`. When the execution completes, the status section of the window title changes from `[Running]` to `[Completed - <Status of Test>]`, and the Execution window dims.

Note If you were not able to see the execution pointer, the tracing option may be disabled. Select **Configure»Station Options** and go to the **Execution** tab. Enable the **Enable Tracing** and **Allow Tracing into Setup/Cleanup** options. Click **OK** to close the Station Options dialog box and select **Execute»Run MainSequence** to rerun your execution.

5. After the execution completes, close the Execution window. You can rerun the sequence by repeating steps 1 through 3.

Tip To reduce the tracing speed, adjust the **Speed** slider control value on the **Execution** tab of the Station Options dialog box.

Running a Sequence Using the Sequential Process Model

In addition to executing a sequence directly, you can execute a sequence using an Execution entry point. As you learned in the *Process Models* section of Chapter 1, *Introduction to TestStand*, an Execution entry point is a sequence in a process model sequence file. When you execute an Execution entry point, it performs a series of operations before and after calling the Main sequence of your sequence file. Common operations of the process model are identifying the UUT, notifying the operator of pass/fail status, generating a test report, and logging results.

Test UUTs Execution Entry Point

The Test UUTs Execution entry point executes the sequence in a loop, prompting for a serial number at the beginning of each loop iteration and appending the results to the test report at the end of each iteration. After the loop is complete, TestStand displays the full report.

Single Pass Execution Entry Point

The Single Pass Execution entry point executes the sequence once without requiring a serial number and then generates a report. Single Pass is useful for debugging your tests and determining that your sequence execution proceeds as you intended.

Complete the following steps to run `MainSequence` in the `Computer.seq` sequence file using the Test UUTs Execution entry point of the Sequential model:

1. Select the Sequential model as your default process model by selecting **Configure»Station Options**.
2. Click the **Model** tab of the Station Options dialog box and select `SequentialModel.seq` from the Station Model ring control.
3. Click **OK**.
4. Select **Execute»Test UUTs**.

   Before executing the steps in `MainSequence`, the process model sequence displays a UUT Information dialog box requesting a serial number.

5. Enter any number and click **OK**.
6. Use the checkboxes in the Test Simulator dialog box to select any test other than the Video or CPU tests to `Fail`. You can also leave all of the tests unchecked to allow the UUT to pass.
7. Click **Done**. Observe the Execution window as the sequence executes.

   After completing the steps in `MainSequence`, the process model displays a banner that indicates the result of the UUT.
8. Click **OK** to close the UUT Result banner.

   The process model now generates a report, which TestStand displays after testing all of the UUTs, and then launches the UUT Information dialog box again.
9. Repeat steps 4 through 8 using several different serial numbers.
10. Click **Stop** to stop the loop and complete the execution.

    After the execution is complete, TestStand displays a report for all of the tested UUTs.
11. Examine the test report and verify that it has indeed recorded the results for each UUT.
12. Close the Execution window.

Running a Sequence Using the Batch Process Model

The Batch process model executes the same test sequence on multiple UUTs at the same time, which means that you start and finish testing all UUTs in unison. The Batch model also provides batch synchronization features. For example, you can specify that because a step applies to a batch as a whole, the step should only run once per batch instead of once for each UUT.

The Batch model also allows you to specify that certain steps or groups of steps cannot run on more than one UUT at a time or that certain steps must run on all UUTs at the same time. The Batch model can also generate batch reports that summarize the test results for all of the UUTs in a batch.

Complete the follow steps to run the `BatchUUT.seq` sequence file using the Test UUTs Execution entry point of the Batch model:

1. Open `<TestStand>\Tutorial\BatchUUT.seq`.

Note You do not need to change your default process model for this tutorial. The sequence file is configured to always use the Batch model, regardless of the default process model of the sequence editor. You can use the Sequence File Properties dialog box to specify that a sequence file always uses a particular process model.

2. Select **Configure»Model Options**.
3. In the Multiple UUT Settings section of the Model Options dialog box, change **Number of Test Sockets** to `2` and enable the **Tile Execution Windows** option.

Tip The **Number of Test Sockets** control specifies the number of UUTs to test in the batch. It is easier to understand the executions if you start with a low number of test sockets.

4. In the Batch Settings section, select **Don't Synchronize** from the Default Batch Synchronization ring control.
5. Select **Execute»Test UUTs**.

   Before executing the steps in `MainSequence`, the process model sequence displays a UUT Information dialog box requesting a batch serial number and UUT serial numbers for each test socket. Notice that you can disable particular test sockets.
6. Enter any batch serial number and UUT serial numbers. Click **Go**.

   After completing the steps in `MainSequence`, the process model displays a banner that indicates the result of the UUTs. This banner allows you to view the batch report and the individual UUT reports.
7. Click **View Batch Report**.

   Notice that you have two viewing options—**Entire File** or **Current Only**. The Entire File view displays all tested batches, while the Current Only view displays the most recently tested batch.
8. Select **Current Only**.

   TestStand launches an *external viewer* to display the reports. By default, HTML and XML reports display in Microsoft Internet Explorer while text reports display in Microsoft Notepad.

Note To configure your system to use another application as an external viewer, select **Configure»External Viewers**. Then, close the Configure External Viewers dialog box and return to the result banner. For more information about using external viewers, refer to the *Using External Report Viewers* section of Chapter 10, *Customizing the Report*.

9. Click **Next Batch**.
10. Repeat steps 6 through 9 for several different batches.
11. Click **Stop to** complete the execution.

    After the execution completes, TestStand displays test reports for all batches and UUTs in the Report tab of the Execution window.

12. Examine the reports and notice that TestStand records results for each batch and UUT.
13. Close the Execution window.

You have completed this tutorial. In the next chapter, you will learn how to add steps to a sequence and edit step properties

3

Editing Steps in a Sequence

In this chapter, you will add a step to a sequence, configure the properties of that step, and add a subsequence call to another sequence.

TestStand contains a set of predefined step types. A *step type* defines a list of standard properties and behaviors and gives you the ability to call code modules using module adapters.

There are a number of predefined step types available in TestStand. For a complete description of the step types listed below, refer to Chapter 4, *Built-In Step Types*, in the *TestStand Reference Manual*.

- Pass/Fail Test
- Numeric Limit Test
- Multiple Numeric Limit Test
- String Value Test
- Action
- Sequence Call
- Statement
- Label
- Goto
- Message Popup
- Call Executable
- Property Loader
- Synchronization (Lock, Rendezvous, Queue, Notification, Wait, Batch Synchronization, Thread Priority, Semaphore, Batch Specification)
- Database (Open Database, Open SQL Statement, Close SQL Statement, Close Database, Data Operation)
- IVI (Dmm, Scope, Fgen, Power Supply, Switch, Tools)

Adding a New Step

In this exercise, you will add a Pass/Fail step to the sequence and configure it to call a function in a LabWindows/CVI DLL code module.

1. Select **File»Open**. Open `Computer.seq` from the `<TestStand>\Tutorial` directory.

   To insert a step that calls a code module, you must specify which module adapter the step uses.

2. Select **LabWindows/CVI** in the Adapter ring control.

   The LabWindows/CVI Adapter allows you to call C functions with a variety of prototypes. The function can be in a dynamic link library (`.dll`), source file (`.c`), object file (`.obj`), or static library file (`.lib`).

   The selected adapter will apply only to the step types that can use the module adapter. The icon for the selected adapter appears as the icon for the step.

3. Right-click the `RAM` step in the Sequence File window and select **Insert Step»Tests»Pass/Fail Test** from the *context menu*. When you make this selection, a Pass/Fail Test step is inserted after the `RAM` step.

   When you insert a step in a sequence, TestStand assigns the adapter you selected in step 3 to that step. If you choose **<None>** as the selected adapter and then insert a step, the step you insert does not call a code module.

Note Once you add a step, you can change the adapter associated with that step in the Step Properties dialog box for the step.

Normally, you use a Pass/Fail Test step to call a code module that makes its own pass/fail determination. After the code module executes, the Pass/Fail Test step evaluates a Boolean expression to determine whether the step passes or fails.

4. By default, the new test is named `Pass/Fail Test`. After you insert the step, the `Pass/Fail Test` step is highlighted. Type `Video Test` and press <**Enter**> to rename the step.

Tip To rename the step at a later time, select the step and press <**F2**> or right-click the step name and select **Rename** from the context menu.

Specifying the Code Module

After you add a new step to the sequence, you must specify the code module that the step executes.

1. Right-click the new `Video Test` step and select **Specify Module** from the context menu.

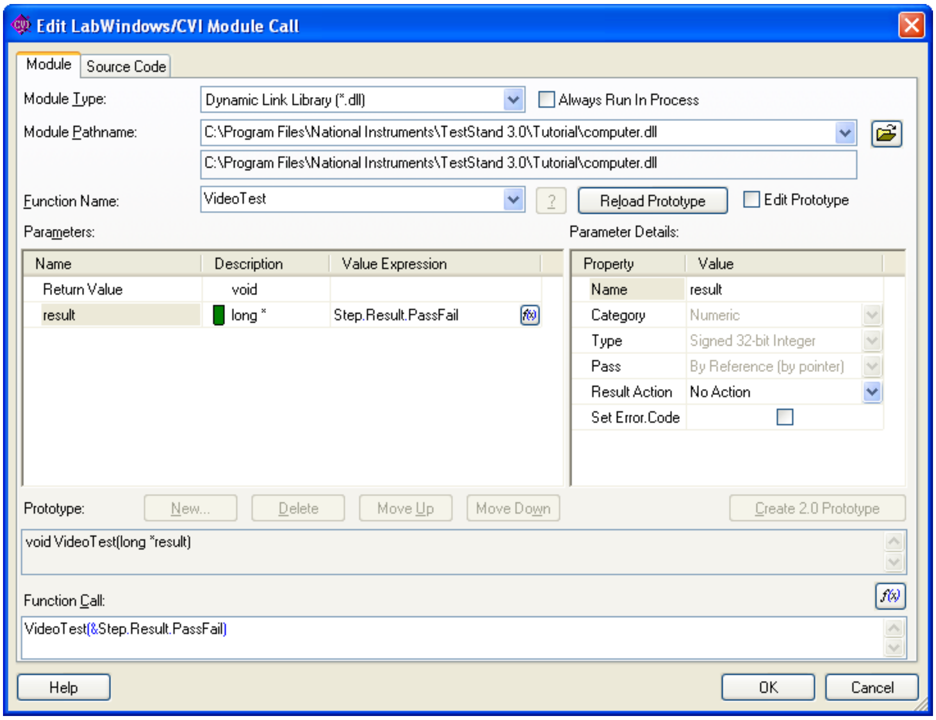
   When you make this selection, the sequence editor launches an Edit Call dialog box specific to the module adapter associated with the step. For the LabWindows/CVI Adapter, the sequence editor displays the Edit LabWindows/CVI Module Call dialog box, which is shown in Figure 3-1.
2. Select **Dynamic Link Library (`*.dll`)** from the Module Type ring control. This selection specifies that the code module for the test calls a function within a DLL.
3. Click the **File Browse** button, which is located to the right of the module pathname control, and select `<TestStand>\Tutorial\computer.dll`. Click **OK**.

Note If prompted, select the **Use a relative path for the file you selected** option.

   When you select a DLL, TestStand attempts to read the type library of the DLL and lists all the exported functions in the Function Name ring control.
4. Select **VideoTest** in the Function Name ring control. TestStand will use the prototype information stored in the type library of the DLL to populate the Parameters table.
5. In the Value Expression column of the **result** parameter, enter the following expression: `Step.Result.PassFail`.

   When TestStand returns from calling the `VideoTest` function, it will insert the value from the **result** parameter into the `Result.PassFail` property of the step.

Figure 3-1 shows the completed Edit LabWindows/CVI Module Call dialog box.


Edit LabWindows/CVI Module Call

Module | Source Code

Module Type: Dynamic Link Library (*.dll) ☐ Always Run In Process

Module Pathname: C:\Program Files\National Instruments\TestStand 3.0\Tutorial\computer.dll

C:\Program Files\National Instruments\TestStand 3.0\Tutorial\computer.dll

Function Name: VideoTest ? Reload Prototype ☐ Edit Prototype

Parameters:

Name	Description	Value Expression
Return Value	void	
result	🟩 long *	Step.Result.PassFail

Parameter Details:

Property	Value
Name	result
Category	Numeric
Type	Signed 32-bit Integer
Pass	By Reference (by pointer)
Result Action	No Action
Set Error.Code	☐

Prototype: New... Delete Move Up Move Down Create 2.0 Prototype

void VideoTest(long *result)

Function Call:

VideoTest(&Step.Result.PassFail)

Help OK Cancel

Figure 3-1. Edit LabWindows/CVI Module Call Dialog Box

6. Click **OK** to close the Edit LabWindows/CVI Module Call dialog box and return to the Sequence File window.

Refer to the *TestStand Help* for more information about the Edit LabWindows/CVI Call Module dialog box. Refer to *Using LabWindows/CVI with TestStand* for more information about calling LabWindows/CVI code modules from TestStand.

Changing Step Properties

Each step in a sequence contains properties. The type of a step determines the set of properties that a step has. All steps have a common set of properties that determine the following attributes:

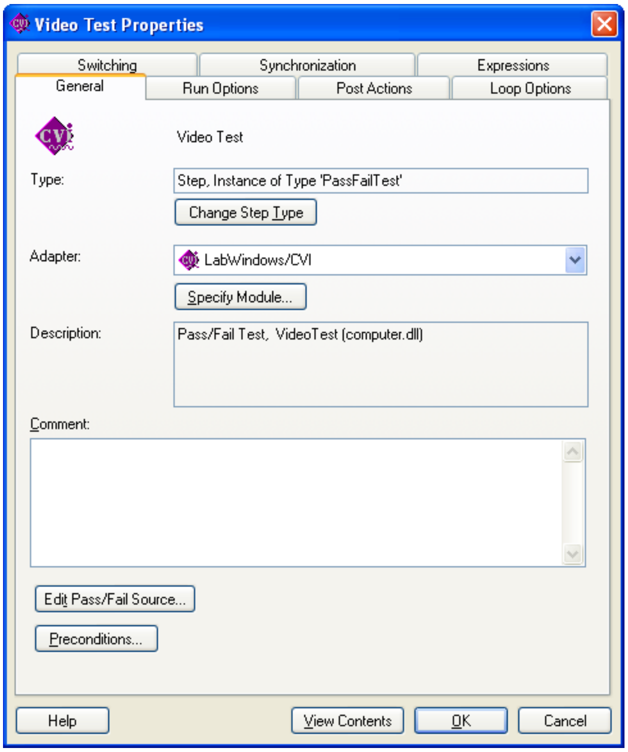
- When to load the step
- When to execute the step
- What information TestStand examines to determine the status of the step
- Whether TestStand executes the step in a loop
- What conditional actions occur upon step completion

You can change the common properties of steps using the tabs in the Step Properties dialog box. In this exercise, you will examine common properties found in the Post Actions and Loop Options tabs, and you will use *preconditions* to modify steps.

Note For detailed information about the Step Properties dialog box, refer to the *TestStand Help*.

1. Right-click the `Video Test` step and select **Properties** from the context menu.

   When you make this selection, TestStand launches the Step Properties dialog box. The title of the Step Properties dialog box is specific to the step that you select. For the Video Test step, the Step Properties dialog box is named Video Test Properties, as shown in Figure 3-2.



Video Test

Type: Step, Instance of Type 'PassFailTest'

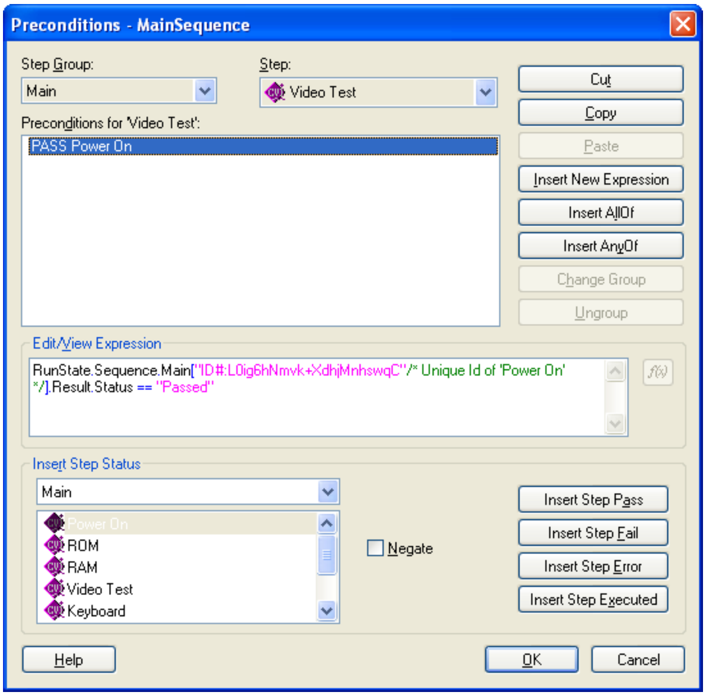
Adapter: LabWindows/CVI

Description: Pass/Fail Test, VideoTest (computer.dll)

Comment:

Figure 3-2. Step Properties Dialog Box

2. Click **Preconditions** in the Step Properties dialog box to launch the Preconditions dialog box, which is shown in Figure 3-3.



Step Group: Main

Step: Video Test

Preconditions for 'Video Test':

PASS Power On

```
RunState.Sequence.Main["ID#:L0ig6hNmvk+XdhjMnhswqC"/* Unique Id of 'Power On'
*/].Result.Status == "Passed"
```

Main

Power On
ROM
RAM
Video Test
Keyboard

Figure 3-3. Preconditions Dialog Box

A precondition specifies the conditions that must be evaluated as `True` for TestStand to execute a step during the normal flow of execution in a sequence, such as only running a step if a previous step passes.

3. For the `Video Test` step, define a precondition so that the step only executes if the `Power On` step passes. Complete the following steps to modify the Preconditions dialog box, as shown in Figure 3-3:

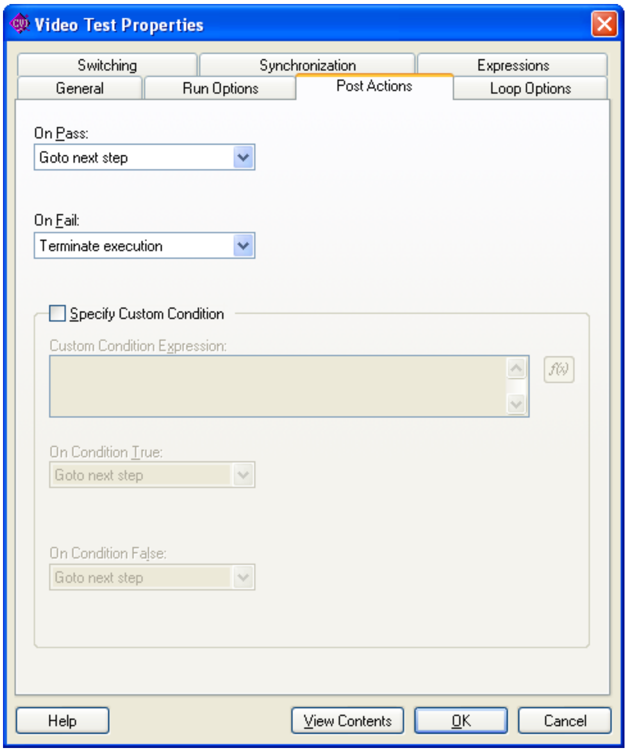
   a. Under the Insert Step Status section, select the `Power On` step from the list of step names for the Main step group.

   b. Click **Insert Step Pass to** add a condition to the precondition list. The Preconditions for 'Video Test' text box now contains the string `PASS Power On`, which indicates that the step executes only if the `Power On` step passes.

   c. Click **OK** to close the Preconditions dialog box and return to the Step Properties dialog box.

4. Click the **Post Actions** tab, which is shown in Figure 3-4. This tab specifies what type of action occurs after the step executes. You can make the action conditional on the Pass/Fail status of the step or on any custom condition expression.

Note For steps that do not have a `Passed` or `Failed` status, you can use a custom condition. to do this, enable the **Specify Custom Condition** option and enter a custom condition expression that evaluates to `True` or `False`. Then, select the appropriate actions in the **On Condition True** and **On Condition False** controls.



On Pass: Goto next step

On Fail: Terminate execution

Specify Custom Condition

Custom Condition Expression:

On Condition True: Goto next step

On Condition False: Goto next step

Figure 3-4. Post Actions Tab

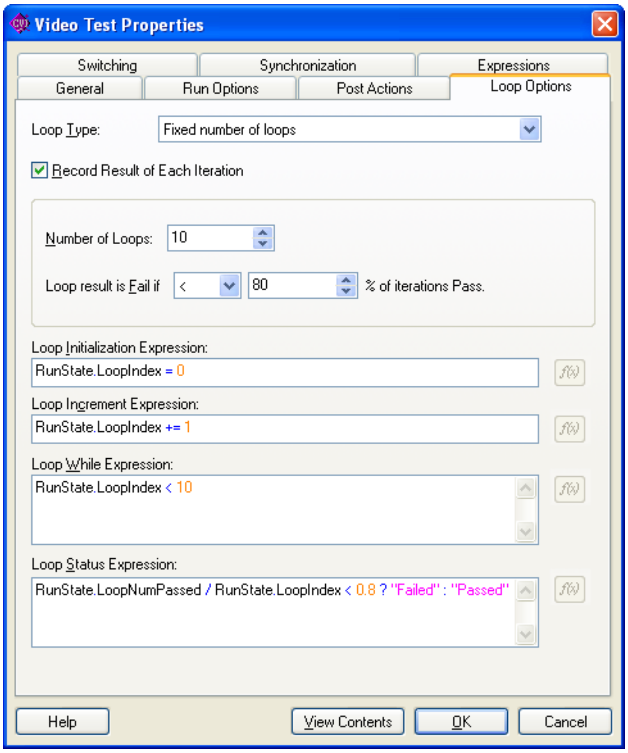
5. Set the On Fail post action to **Terminate execution**.
6. Click the **Loop Options** tab, which is shown in Figure 3-5. Use this tab to configure an individual step to run repeatedly in a loop when it executes.

   Use the Loop Type ring control to select the type of looping for the step. TestStand determines the final status of the step based on whether a specific number of passes or failures occur, or the number of loop iterations reaches the maximum.

7. Change the following control values in the **Loop Options** tab:

Loop Type	Fixed number of loops
Number of Loops	`10`
Loop result is Fail if	`< 80` %

   Using these settings, TestStand executes the Video Test step ten times and sets the overall status for the step to `Failed` if less than eight of the ten iterations pass. Figure 3-5 shows the completed Loop Options tab.



Loop Type: Fixed number of loops

[✓] Record Result of Each Iteration

Number of Loops: 10

Loop result is Fail if < 80 % of iterations Pass.

Loop Initialization Expression:
```
RunState.LoopIndex = 0
```
Loop Increment Expression:
```
RunState.LoopIndex += 1
```
Loop While Expression:
```
RunState.LoopIndex < 10
```
Loop Status Expression:
```
RunState.LoopNumPassed / RunState.LoopIndex < 0.8 ? "Failed" : "Passed"
```

Figure 3-5. Loop Options Tab

8. Click **OK** to close the Step Properties dialog box.

   Notice that the Execution Flow column in the Sequence File window shows that the Video test step contains Preconditions, Looping, and Post Actions.

9. Select **File»Save As**. Save the sequence in the `<TestStand>\Tutorial` directory as `Computer2.seq`.

Tip If other people will be using the tutorial files on your computer, use the **Save As** option to save your files with non-default file names. When you must save a file, this manual specifies the suggested name.

10. Run the sequence by selecting **Execute»Single Pass**. Notice that if you select the Video test step to fail, the sequence immediately terminates after calling the Video test step ten times in a loop.

    After TestStand generates the report, notice that when the Video test step executes, the result of each loop iteration is recorded in the report.
11. Close the Execution window.

Calling a Subsequence from a Sequence

In TestStand, you can call another sequence using a Sequence Call step. You can locate your called sequence within the calling sequence file or a separate sequence file. In this tutorial, you will add a Sequence Call step to your current sequence.

1. Right-click the `Power On` step and select **Insert Step» Sequence Call** from the context menu. When you make this selection, the sequence editor inserts a Sequence Call step after the `Power On` step.
2. Rename the step `CPU Test`.
3. Specify which sequence the step invokes, as follows:
    a. Right-click the `CPU Test` step.
    b. Select **Specify Module** from the context menu, which launches the Edit Sequence Call dialog box.
    c. Click the **File Browse** button, which is located to the right of the File Pathname control.
    d. Select `CPU.seq` from the `<TestStand>\Tutorial` directory. The completed Edit Sequence Call dialog box is shown in Figure 3-6.

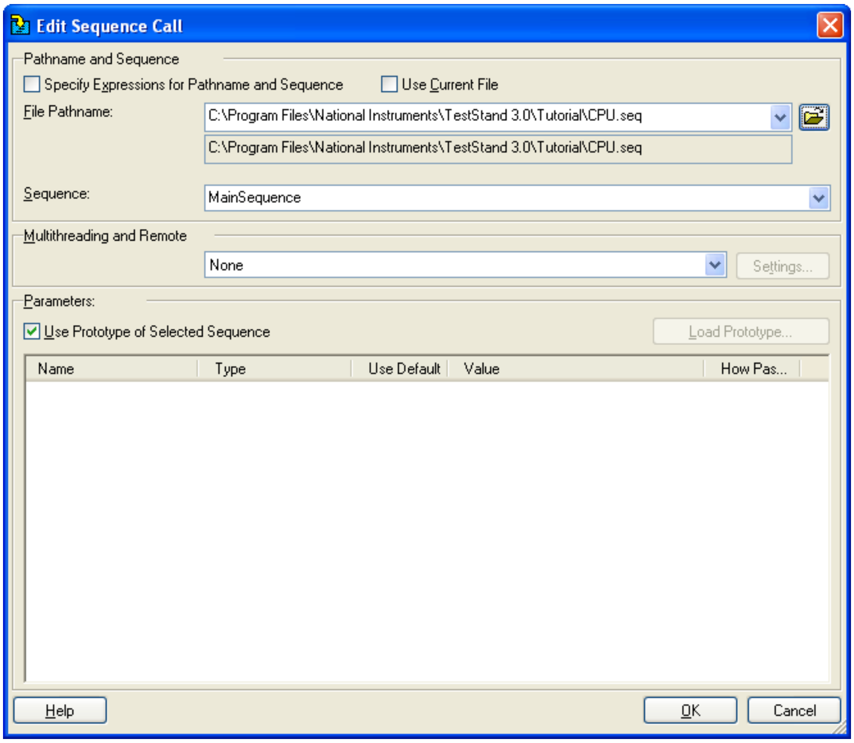

Edit Sequence Call
Pathname and Sequence
Specify Expressions for Pathname and Sequence
Use Current File
File Pathname:
C:\Program Files\National Instruments\TestStand 3.0\Tutorial\CPU.seq
C:\Program Files\National Instruments\TestStand 3.0\Tutorial\CPU.seq
Sequence:
MainSequence
Multithreading and Remote
None
Settings...
Parameters:
Use Prototype of Selected Sequence
Load Prototype...
Name
Type
Use Default
Value
How Pas...
Help
OK
Cancel

Figure 3-6. Edit Sequence Call Dialog Box

   e. Click **OK** to close the Edit Sequence Call dialog box.

4. Select **File»Save to** save your changes to the sequence file.
5. Right-click the `CPU Test` step again and select **Open Sequence** from the context menu. When you make this selection, the sequence editor opens `CPU.seq` and displays its `MainSequence`.
6. Select **Execute»Run MainSequence**. Examine the execution of this sequence.
7. Close the Execution window.
8. Close the `CPU.seq` Sequence File window.
9. Select **Execute»Single Pass**.
10. Select a test to fail and click **Done**.

    After the sequence executes, examine the test report and notice that TestStand logs the results of the steps in the subsequence along with the steps from the *parent sequence*.
11. Close the Execution window.

Refer to the *TestStand Help* for more information about the Edit Sequence Call dialog box. Refer to Chapter 5, *Module Adapters*, in the *TestStand Reference Manual* for more information about calling sequences.

You have completed this tutorial. In the next chapter, you will learn how to use TestStand's executing and debugging tools.

4 Debugging Sequences

TestStand has several features you can use to debug a sequence, including tracing, breakpoints, conditional breakpoints, single-stepping, and watch expressions. In this chapter, you will use the sequence debugging features of TestStand to single-step through your sequence during an execution. Refer to the *TestStand Help* for more information about debugging commands.

Step Mode Execution

In this tutorial, you will learn how to run your sequences in step mode execution.

Note Before beginning this tutorial, verify in the Station Options dialog box that the Enable Tracing and Tracing Into Setup/Cleanup options are enabled.

1. Select **File»Open** and select `<TestStand>\Tutorial\Computer2.seq`, which you created in Chapter 3, *Editing Steps in a Sequence*. You can also find this file in the `<TestStand>\Tutorial\Solution` directory.
2. Select **Execute»Break At First Step**. This command suspends an execution on the first step that TestStand executes. When enabled, this command has a checkmark beside it in the **Execute** menu.
3. Select the **Cleanup** tab in the Sequence File window.

   TestStand executes the Cleanup step group after the Main step group executes, regardless of whether the sequence completes successfully. If a step in the Setup or Main step groups causes a *run-time error* to occur or if the operator terminates the execution, the flow of execution stops and jumps to the Cleanup step group.
4. Add a step that displays a message popup to the Cleanup step group to demonstrate this behavior, as follows:
   a. Right-click in the empty step list of the Cleanup step group and select **Insert Step»Message Popup** from the context menu.
   b. Rename the step `Cleanup Message`.

c. Right-click the `Cleanup Message` step and select **Edit Message Settings** from the context menu. When you make this selection, the sequence editor displays the Configure Message Popup Step dialog box.

d. Enter the text `"Cleanup Message"`, including the quotation marks, in the **Title Expression** expression control.

e. Enter the text `"I am now in the Cleanup Step Group."`, including the quotation marks, in the **Message Expression** expression control.

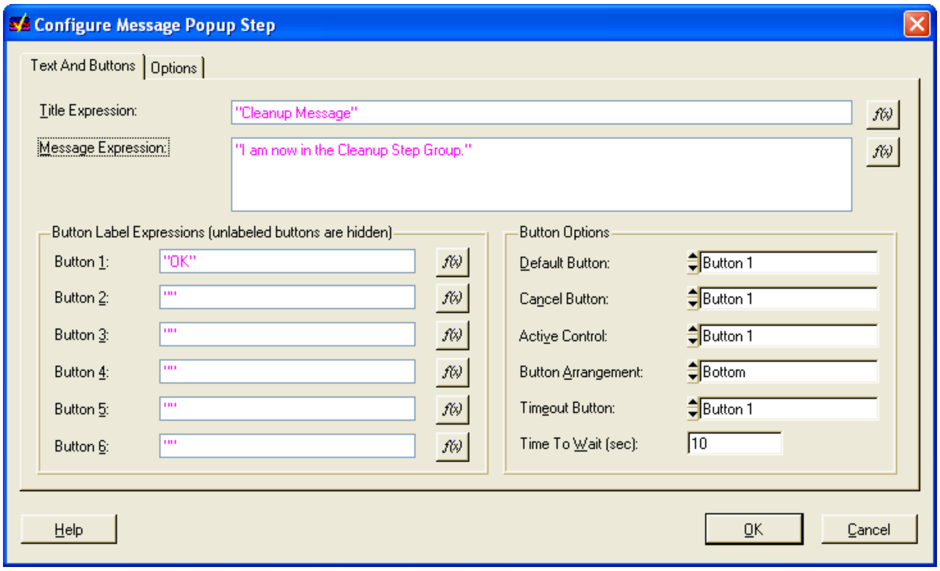
Note You must enclose literal strings in double quotation marks (`"..."`) in any TestStand expression field.

f. In the Button Options section, select **Button 1** in the **Timeout Button** control. The Timeout Button control specifies which message box button activates automatically after a timeout period expires.

g. Enter `10` into the **Time to Wait (sec)** control.

When the sequence is executed, the Cleanup Message step displays a notification message that automatically dismisses itself if the user does not make an entry within ten seconds. This is useful if an operator is not present to acknowledge a non-critical message that displays during testing.

Figure 4-1 shows the Text And Buttons tab of the completed Configure Message Popup Step dialog box.


Configure Message Popup Step

Text And Buttons | Options

Title Expression: "Cleanup Message"

Message Expression: "I am now in the Cleanup Step Group."

Button Label Expressions (unlabeled buttons are hidden)

Button 1:	"OK"
Button 2:	""
Button 3:	""
Button 4:	""
Button 5:	""
Button 6:	""

Button Options

Default Button:	Button 1
Cancel Button:	Button 1
Active Control:	Button 1
Button Arrangement:	Bottom
Timeout Button:	Button 1
Time To Wait (sec):	10

Figure 4-1. Configure Message Popup Step Dialog Box

Note The Options tab of the Configure Message Popup Step dialog box allows you to enable a response text box for operator input, position the message popup using coordinates, and make the message popup a *modal dialog box* with respect to the application. For this example, leave all settings on the Options tab set to their default values. Refer to Chapter 4, *Built-In Step Types*, in the *TestStand Reference Manual* for more information about the Message Popup step type. Refer to the *TestStand Help* for more information about the Options tab.

h. Click **OK** to close the Configure Message Popup Step dialog box.

5. Select **File»Save As**. Save the sequence as `Computer3.seq` in the `<TestStand>\Tutorial` directory.
6. Select **Execute»Run MainSequence**.

   After the execution starts, the sequence editor immediately pauses at the first step of the sequence. This behavior is caused by the **Break At First Step** option, which you enabled in step 2 of this tutorial. Figure 4-2 illustrates the sequence editor at this point in the tutorial.

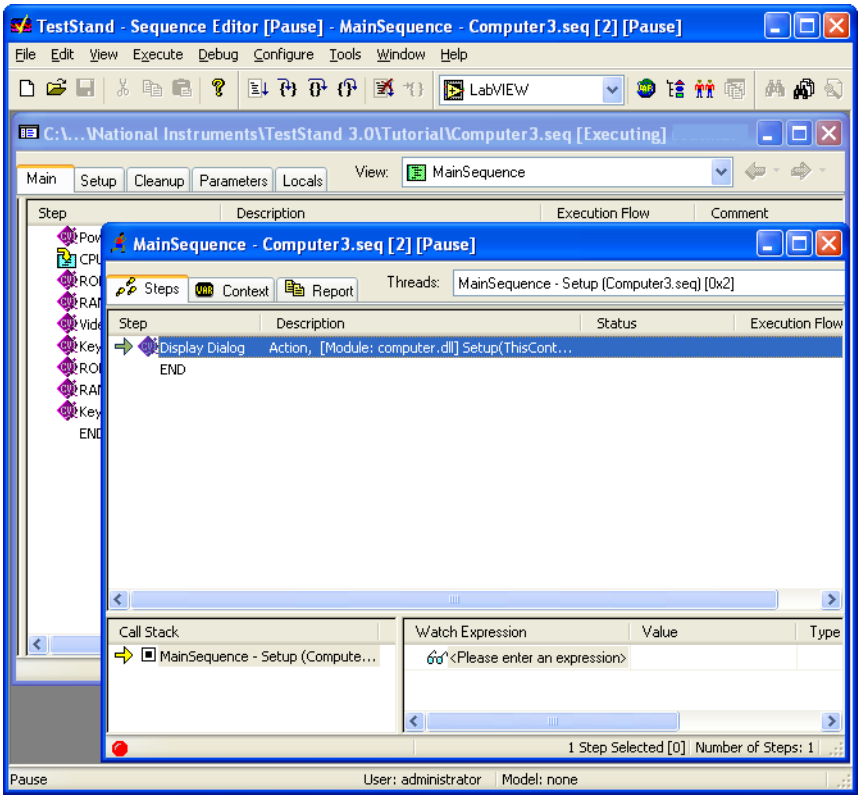
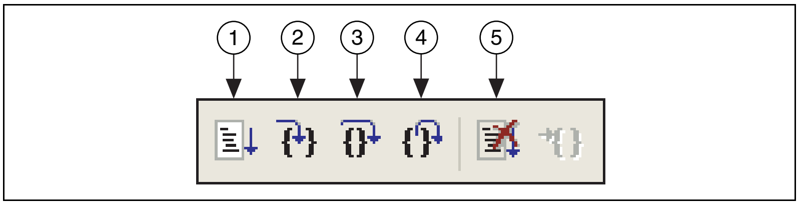

TestStand - Sequence Editor [Pause] - MainSequence - Computer3.seq [2] [Pause]
File Edit View Execute Debug Configure Tools Window Help
LabVIEW
C:\...\National Instruments\TestStand 3.0\Tutorial\Computer3.seq [Executing]
Main Setup Cleanup Parameters Locals
View: MainSequence
Step Description Execution Flow Comment
MainSequence - Computer3.seq [2] [Pause]
Steps Context Report
Threads: MainSequence - Setup (Computer3.seq) [0x2]
Step Description Status Execution Flow
Display Dialog Action, [Module: computer.dll] Setup(ThisCont...
END
Call Stack
MainSequence - Setup (Compute...
Watch Expression Value Type
<Please enter an expression>
1 Step Selected [0] Number of Steps: 1
Pause User: administrator Model: none

Figure 4-2. Paused Execution of Computer3.seq

Notice the left corner of the status bar in the Execution window, which contains an LED indicator that displays the running state of the execution. The running state should now be paused, as indicated by the red LED.

When the execution is paused, you can single-step through the sequence using the Step Into, Step Over, and Step Out commands in the Debug section of the toolbar, as shown in Figure 4-3. These tools can also be found in the Debug menu of the sequence editor. For a detailed discussion of the single-stepping tools, refer to Chapter 3, *Executions*, in the *TestStand Reference Manual*.



1	Run	4	Step Out
2	Step Into	5	Terminate Execution
3	Step Over		

Figure 4-3. Single-Stepping Toolbar Buttons

7. Click **Step Over** to execute the Display Dialog step. This step displays the Test Simulator dialog box.
8. Select the RAM test to fail, and click **Done**.

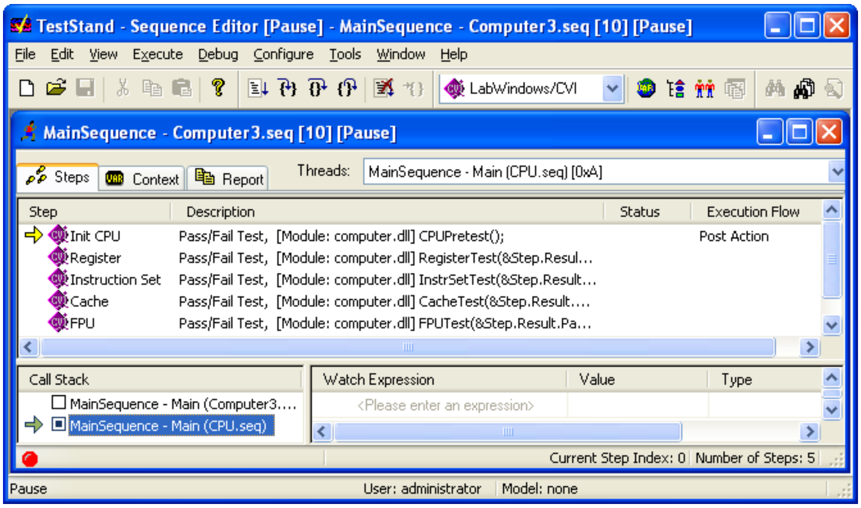
   After you close the Test Simulator dialog box, the sequence editor suspends the sequence execution at the end of the Setup step group on END.
9. Activate the Sequence File window for `Computer3.seq` by selecting it from the **Window** menu.
10. Right-click the `CPU Test` step in the Main step group, and select **Breakpoint»Toggle Breakpoint** from the context menu. Notice that a dark red stop sign icon appears to the left of the step name.

Note You can right-click the stop sign icon to bring up the Breakpoint Settings dialog box. Use this dialog box to specify an expression that must be `True` in order for the breakpoint to be valid. These breakpoints are conditional breakpoints and are identified with a bright red stop sign icon. For more information about expressions, refer to Chapter 5, *Using Variables and Properties*.

11. Return to the Execution window by selecting it from the **Window** menu.
12. Select **Debug»Resume** to continue the execution. After you make this selection, the sequence editor suspends the execution on the CPU Test step again.
13. Click **Step Into** to step into the subsequence.

Tip You may have to adjust the size of the Execution window to make all elements visible.

Figure 4-4 shows the Steps view for the Execution window after you step into the subsequence.



TestStand - Sequence Editor [Pause] - MainSequence - Computer3.seq [10] [Pause]

File Edit View Execute Debug Configure Tools Window Help

LabWindows/CVI

MainSequence - Computer3.seq [10] [Pause]

Steps | Context | Report — Threads: MainSequence - Main (CPU.seq) [0xA]

Step	Description	Status	Execution Flow
Init CPU	Pass/Fail Test, [Module: computer.dll] CPUPretest();		Post Action
Register	Pass/Fail Test, [Module: computer.dll] RegisterTest(&Step.Resul…		
Instruction Set	Pass/Fail Test, [Module: computer.dll] InstrSetTest(&Step.Result…		
Cache	Pass/Fail Test, [Module: computer.dll] CacheTest(&Step.Result….		
FPU	Pass/Fail Test, [Module: computer.dll] FPUTest(&Step.Result.Pa…		

Call Stack
MainSequence - Main (Computer3.…
MainSequence - Main (CPU.seq)

Watch Expression	Value	Type
<Please enter an expression>		

Current Step Index: 0 Number of Steps: 5

Pause User: administrator Model: none

Figure 4-4. Steps View while Suspended in Subsequence

Usually, when a step invokes a subsequence, the sequence that contains the calling step waits for the subsequence to return. The subsequence invocation is *nested* in the invocation of the calling sequence. The chain of active sequences that are waiting for nested subsequences to complete is called the *call stack*. The last item in the call stack is the most nested sequence invocation.

The Call Stack pane in the lower left half of the Execution window displays the call stack for the execution. A yellow pointer icon appears to the left of the most nested sequence invocation. The call stack in Figure 4-4 shows that `MainSequence` in `Computer3.seq` is calling `MainSequence` in `CPU.seq`.

When the execution suspends, you can view a sequence invocation in the call stack by clicking its radio button.

14. Click each radio button in the Call Stack pane to view the status of each sequence invocation. Return to the most nested sequence invocation in the call stack.
15. Click **Step Over to** start stepping through the subsequence.
16. Before reaching the end of the sequence, click **Step Out**. TestStand resumes the execution through the end of the current sequence and suspends the execution at the next step or breakpoint, whichever comes first.

17. Continue single-stepping through the sequence by clicking **Step Over** until the execution completes. The last step executed is the Cleanup Message step that you added to the Cleanup step group.
18. Click **OK** to close the Cleanup Message dialog box before completing the execution. The Execution window dims when the execution is complete.

Note Do not close the Execution window.

19. Rerun the execution by selecting **Execute»Restart**. The Execution window must be the active window to restart the sequence.
20. After the sequence editor suspends the execution on the first step, select **Debug»Terminate**.

Note TestStand displays the Cleanup Message dialog box even though you terminated the sequence execution. When an operator or run-time error terminates the execution, TestStand proceeds immediately to the steps in the Cleanup step group.

21. Click **OK** to close the Cleanup Message dialog box.
22. Rerun the execution again by selecting **Execute»Restart**.
23. After the sequence editor suspends the execution on the first step, select **Debug»Abort**. Notice that the execution of the sequence immediately stops, and TestStand does not execute any steps in the Cleanup step group.
24. Close the Execution window.
25. Save the sequence by selecting **File»Save**.
26. Close the `Computer3.seq` window.

You have completed this tutorial. In the next chapter, you will learn how to create and use TestStand variables and properties.

5 Using Variables and Properties

This chapter teaches you how to create and use variables and properties in TestStand, and points out features that help you monitor the values of those variables and properties.

In TestStand, you can define variables with various scopes to share data between steps of a sequence or even between several sequences. You can define variables that are local to a sequence, global to a sequence file, and global to the test station. Use these types of variables as follows:

- Use *local variables* to store data relevant to the execution of the sequence. Each step in the sequence can directly access sequence local variables.
- Use *sequence file global variables* to store data relevant to the entire sequence file. Each sequence and step in the sequence file can directly access these global variables.
- Use *station global variables* to maintain statistics or to represent the configuration of your test station.You can access station global variables from any sequence or step on that test station. Unlike other variables, the values of station global variables are saved from one TestStand session to the next.

Using Local Variables

In this tutorial, you will learn how to create and use local variables. You can also apply the concepts that you learn in this tutorial to sequence file global and station global variables.

1. Select **File»Open** and then select `<TestStand>\Tutorial\Computer2.seq`, which you created in Chapter 3, *Editing Steps in a Sequence*. You can also find this file in the `<TestStand>\Tutorial\Solution` directory.
2. Display the Main sequence in the Sequence File window by selecting `MainSequence` in the View ring control.

3. Click the **Locals** tab of the Sequence File window. When you make this selection, the view displays all of the local variables currently defined for `MainSequence` in `Computer2.seq`.

   By default, TestStand only defines one local variable when creating a new sequence. TestStand uses the ResultList local variable, of type array, to store the results from the steps it executes in this sequence. The array of step results is used in report generation and for logging results to a database.

4. Right-click in the right pane and select **Insert Local»Number** from the context menu. When you make this selection, the sequence editor inserts a new numeric local variable.

5. Rename the local variable `LoopIndex`.

6. Add steps to the sequence to make it loop on a set of steps based on the value of the `LoopIndex` local variable, as follows:

   a. Click the **Main** tab in the Sequence File window to display the steps in the Main step group.

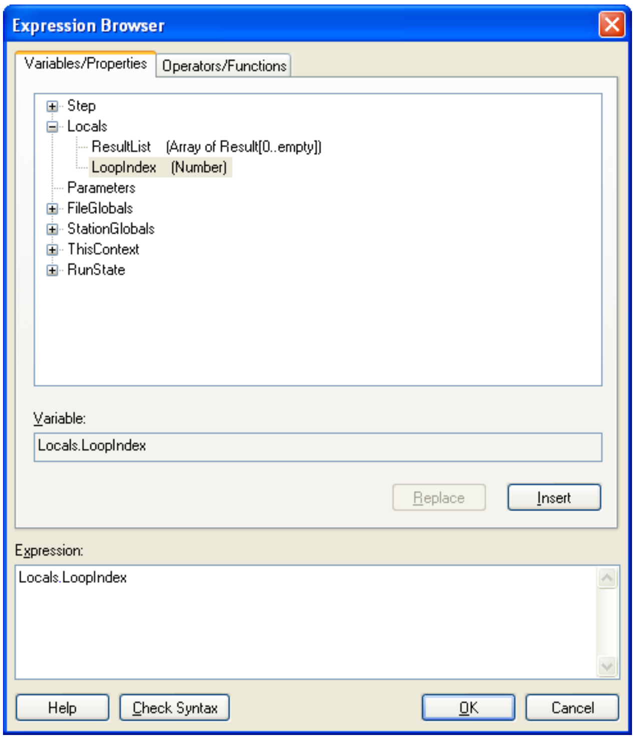
   b. Right-click the `Power On Test` step and select **Insert Step»Statement** from the context menu.

      Use Statement steps to execute expressions that TestStand evaluates when it executes the step. For example, you can use a Statement step to increment the value of a local variable in the sequence file.

   c. Rename the new step `Reset Loop Index`.

   d. Right-click the `Reset Loop Index` step and select **Edit Expression** from the context menu. This command opens the Edit Statement Step dialog box.

   e. Click the **Expression Browse** button to open the Expression Browser dialog box, which is shown in Figure 5-1.



Variable: Locals.LoopIndex

Expression: Locals.LoopIndex

Figure 5-1. Expression Browser Dialog Box

Use the Expression Browser to interactively build an expression and to create variables and parameters. The Expression Browser contains two tabs: Variables/Properties and Operators/Functions. You can select variables and properties from the tree view on the Variables/Properties tab. The Operators/Functions tab contains a list of all predefined operators and functions.

The Expression Browser contains help text for the currently selected operator or function. TestStand supports all applicable expression operators and syntax that you use in C, C++, Java, and Visual Basic. For additional information about supported syntax, refer to the *TestStand Help*.

Note All expression fields in TestStand provide editing help through drop-down lists and context-sensitive highlighting. At any point while editing an expression, you can select <Ctrl-Space> to get a drop-down list of valid expression elements.

You can create variables directly from the Expression Browser using the context menu.

f. Hold your mouse pointer over properties shown in the Expression Browser to get tip strips.

   The tip strip displays **Right-click to insert new variable** for those properties under which you can create a variable.

   – Expand the `Locals` item by double-clicking the name or by clicking the plus icon in front of the item. When you expand a tree view item, the Expression Browser displays all the items under the base item. Each item in the tree view is a property or variable of TestStand.

   – Select the `LoopIndex` variable under the `Locals` property and click **Insert**. When you make this selection, the Expression Browser enters `Locals.LoopIndex` into the Expression control.

Note To refer to a subproperty, use a period to separate the name of the property from the name of the subproperty. For example, reference the LoopIndex subproperty of the Locals property as `Locals.LoopIndex`.

   – Click the **Operators/Functions** tab, and select the **Assignmen**t category from the left pane.

   – Select the assignment operator (**=**) from the right pane.

   – Click **Insert** to add the assignment operator to the expression. You should now see `Locals.LoopIndex =` in the Expression control.

   – Place the text cursor immediately after the equals sign in the expression control and then enter `0`. The expression now reads `Locals.LoopIndex =0`.

   – Click **Check Syntax** to verify that the expression does not contain any illegal syntax.

   – Click **OK to** return to the Edit Statement Step dialog box.

g. Click **OK** to close the Edit Statement Step dialog box and return to the Sequence File window.

h. Right-click the `Reset Loop Index` step and select **Insert Step»Label** from the context menu.

   Label steps are normally used as targets for Goto steps, as you will see in Step 8. By using a Label step, you can rearrange or delete other steps around the Label step without having to change the target step that the Goto step references.

i. Rename the new step `Loop Begin`.

7. Add a **Statement** step to increment the value of the `LoopIndex` local variable, as follows:
   a. Right-click the `RAM Test` step and select **Insert Step»Statement** from the context menu.
   b. Rename the new step `Increment Loop Index`.
   c. Right-click the `Increment Loop Index` step and select **Edit Expression** from the context menu.
   d. Click the **Expression Browse** button to launch the Expression Browser.
   e. Use the Expression Browser to build the following expression, or enter it directly into the Expression control:

      ```
      Locals.LoopIndex ++
      ```

Tip The increment operator (++) is located in the **Arithmetic** group of the **Operators/Functions** tab.

   f. Click **OK** twice to close both the Expression Browser and the Edit Statement Step dialog boxes.

8. Add a **Goto** step to the loop structure in the sequence, as follows:
   a. Right-click the `Increment Loop Index` step and select **Insert Step»Goto** from the context menu.
   b. Rename the step `Loop End`.
   c. Right-click the `Loop End` step and select **Edit Destination** from the context menu.
   d. Select the `Loop Begin` step using the Destination ring control.
   e. Click **OK** to close the Edit Goto Step dialog box.

9. To complete the loop structure, set a precondition for the `Loop End` step so that it only executes if the value of the `LoopIndex` variable is below a certain value, as follows:
   a. Right-click the `Loop End` step and select **Preconditions** from the context menu to open the Preconditions dialog box.
   b. Click **Insert New Expression**.

c. Click the **Expression Browse** button in the Edit/View Expression section of the Preconditions dialog box to open the Expression Browser.

d. Using the Expression Browser, create the following expression:

```
Locals.LoopIndex < 5
```

Tip The less than operator (**<**) is located in the **Comparison** group on the **Operators/Functions** tab.

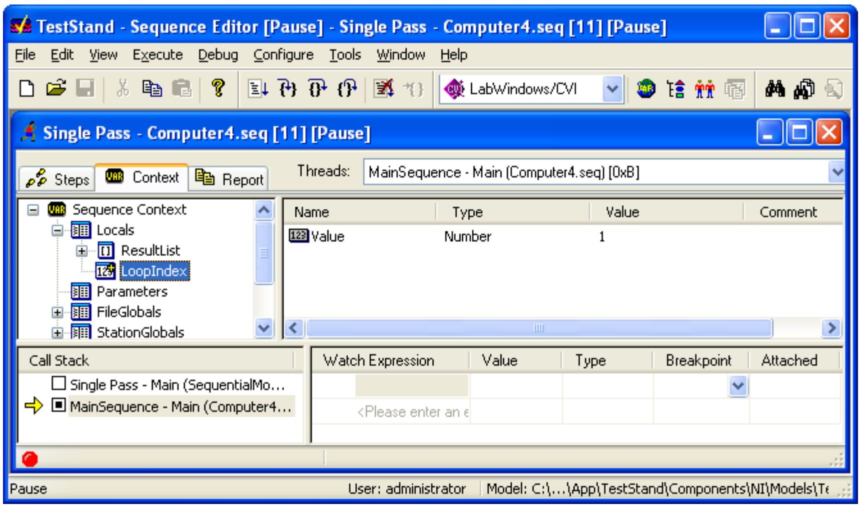
e. Click **OK** twice to close both the Expression Browser and the Preconditions dialog box.

10. Select **File»Save As**. Save the sequence as `Computer4.seq` in the `<TestStand>\Tutorial` directory.
11. Select the **Execute** menu to see if the **Break At First Step** option is enabled. If it is enabled, disable it.
12. Run the sequence by selecting **Execute»Single Pass**.
13. Click **Done** in the Test Simulator dialog box.
14. After the sequence executes, examine the test report and notice that TestStand executed the steps within the loop (`CPU`, `ROM`, and `RAM`) five times.
15. Close the Execution window.

Using the Context Tab

In this tutorial, you will use the Context tab of the Execution window to examine the value of the LoopIndex variable while TestStand executes the sequence.

1. Right-click the `Loop End` step and select **Breakpoint»Toggle Breakpoint** from the context menu to set a breakpoint on the step. In the Sequence File window, a dark red stop sign icon appears beside the `Loop End` step indicating this breakpoint.
2. Run the sequence by selecting **Execute»Single Pass**.
3. Click **Done** in the Test Simulator dialog box. The execution suspends on the `Loop End` step.
4. Select the **Context** tab of the Execution window.
5. Expand the `Locals` property in the upper left tree view pane.
6. Click the `LoopIndex` property under the `Locals` property. Notice that the numeric value of `LoopIndex` is `1`, as shown in Figure 5-2.


TestStand - Sequence Editor [Pause] - Single Pass - Computer4.seq [11] [Pause]
File Edit View Execute Debug Configure Tools Window Help
LabWindows/CVI
Single Pass - Computer4.seq [11] [Pause]
Steps Context Report
Threads: MainSequence - Main (Computer4.seq) [0xB]
Sequence Context
Locals
ResultList
LoopIndex
Parameters
FileGlobals
StationGlobals
Call Stack
Single Pass - Main (SequentialMo…
MainSequence - Main (Computer4…
Watch Expression
Value
Type
Breakpoint
Attached
<Please enter an e
Pause
User: administrator
Model: C:\...\App\TestStand\Components\NI\Models\Te

Name	Type	Value	Comment
Value	Number	1	

Figure 5-2. Context Tab

Before executing the steps in a sequence, TestStand creates a run-time copy of the sequence. This allows parallel executions of the same sequence to run such that each execution does not alter variable or property values in other executions. When an execution completes, TestStand discards the run-time sequence copy.

TestStand maintains a *sequence context* for each active sequence. The sequence context represents the execution state of the sequence. The contents of the sequence context change depending on the currently executing sequence and step. Both the run-time copies and the original versions of properties and variables are accessible from a sequence context.

The Context tab displays the sequence context for the sequence invocation that is currently selected in the Call Stack pane. The sequence context contains all the variables and properties that the steps in the selected sequence invocation can access. Use the Context tab to examine and modify the values of these variables and properties.

Table 5-1 lists the first-level properties in the sequence context and describes their contents. Refer to the *TestStand Help* and to Chapter 2, *Sequence Files and Workspaces*, in the *TestStand Reference Manual* for more information about sequence contexts.

Table 5-1. First-Level Properties of the Sequence Context

Sequence Context Subproperty	Description
`Step`	Contains the properties of the currently executing step in the current sequence invocation. The Step property only exists while a step executes. It does not exist when the execution is between steps, such as at a breakpoint.
`Locals`	Contains the sequence local variables for the current sequence invocation.
`Parameters`	Contains the sequence parameters for the current sequence invocation.
`FileGlobals`	Contains the sequence file global variables for the current execution.
`StationGlobals`	Contains the station global variables for the engine invocation. TestStand maintains a single copy of the station global variables in memory.
`ThisContext`	Holds a reference to the current sequence context. Use this property to pass the entire sequence context as an argument to a subsequence or a code module.
`RunState`	Contains properties that describe the state of execution in the sequence invocation, such as the current step, the current sequence, and the calling sequence.

7. Select **Debug»Resume** and notice that the execution resumes and suspends at the `Loop End` step again.
8. Click the **Context** tab again and notice that the value of `Locals.LoopIndex` is now `2`. Leave the execution in the paused state in preparation for the next tutorial.

Using the Watch Expression Pane

In this tutorial, you will monitor the value of the LoopIndex variable using the Watch Expression pane. The Watch Expression pane is located in the lower right corner of the Execution window. The Watch Expression pane displays the values of watch expressions you enter. TestStand updates the values in the Watch Expression pane when execution suspends at a breakpoint. If tracing is enabled, TestStand also updates the values after executing each step.

Normally, you enter watch expressions to monitor the values of variables and properties as you trace or single-step through a sequence. You can drag individual variables or properties from the Context tab to the Watch Expression pane.

To create a watch expression for the LoopIndex property, complete the following steps:

1. Click the `LoopIndex` property in the tree view of the **Context** tab and drag the property to the Watch Expression pane.

   Notice that the value of the LoopIndex watch expression immediately updates to `2`.
2. Select **Debug»Resume** and notice that when the execution suspends on the `Loop End` step again, the value of the `Locals.LoopIndex` watch expression changes from `2` to `3`.
3. Remove the breakpoint by clicking the breakpoint icon to the left of the `Loop End` step.
4. Select **Debug»Resume** to complete the execution.
5. Close the Execution window.

You can create more complex expressions in the Watch Expression pane. to add a new expression, right-click in the Watch Expression pane and select **Add** from the context menu. to edit an existing watch expression, right-click the expression and select **Edit** from the context menu. Both of these selections display the Watch Expression Settings dialog box, which you use to create a watch expression. You can also edit the existing watch expression directly in the Watch Expression column of the Watch Expression pane.

Note You can also copy watch expressions and paste them into the Watch Expression pane of a subsequent execution. Watch expressions are automatically maintained in subsequent executions if you select **Execution»Restart** after an execution has completed but before you close the Execution window. If you enable the Persist Breakpoints and Watch Expressions option in the Preferences tab of the Station Options dialog box, watch expressions are automatically maintained in all subsequent executions.

For more information about the Watch Expression pane, refer to the *TestStand Help*. For more details about using variables and properties in TestStand, refer to Chapter 3, *Executions*, and Chapter 8, *Customizing and Configuring TestStand*, in the *TestStand Reference Manual*. You can also refer to the following *TestStand Help* topics for information about using variables and properties in TestStand: *Object Relationships* and *Sequence Context Properties*.

You have completed this tutorial. In the next chapter, you will learn how to use callbacks in TestStand.

6 Using Callbacks

This chapter describes how to customize the execution of a sequence within TestStand using callbacks. A *callback* is a sequence that is used to handle common tasks such as serial number inquiry or report logging.

All of the default callback sequences included in TestStand are provided in source form so that you can edit or replace them to customize TestStand for your particular application. In this tutorial session, you will replace one of the default TestStand callbacks with your own.

Process Model Callbacks

TestStand process models contain sequences that define the operations TestStand performs before and after it tests a UUT. If you want to invoke a sequence in one of the process models, you can run one of the entry point sequences in the models. Each model uses the default Execution entry points, Test UUTs and Single Pass, as discussed in the *Running a Sequence Using the Sequential Process Model* section of Chapter 2, *Loading and Running Sequences*. The process models also contain *Model callbacks*, which are sequences that allow you to customize the behavior of a process model for each Main sequence that uses it without forcing you to edit the process model directly.

For example, the TestStand process models define a TestReport Callback that generates the test report for each UUT. Normally, the TestReport Callback in the process model file is sufficient because it handles many types of test results. You can, however, override the default TestReport Callback by defining a different TestReport Callback in a particular *client sequence file*. to alter the behavior of the process models for all sequences, you can modify the process models or replace the models entirely.

Note If you want to modify a TestStand process model directly, copy the `TestStandModels` directory to a new subdirectory under the `<TestStand>\Components\User\Models` directory. By placing your modifications under `<TestStand>\Components\User`, you ensure that a newer installation of TestStand does not overwrite your customizations.

Execution entry points in process models use callbacks to invoke the Main sequence in a *client sequence file*. Each client sequence file must define a sequence by the name of `MainSequence`. The process models contain a MainSequence Callback that serves as a place holder. The MainSequence Callback in the client sequence file overrides the MainSequence Callback place holder in the process model file.

Figure 6-1 shows the callbacks that are called by the default TestStand process model, the Sequential model, and the order in which TestStand executes the callbacks within the Test UUTs and Single Pass Execution entry points.

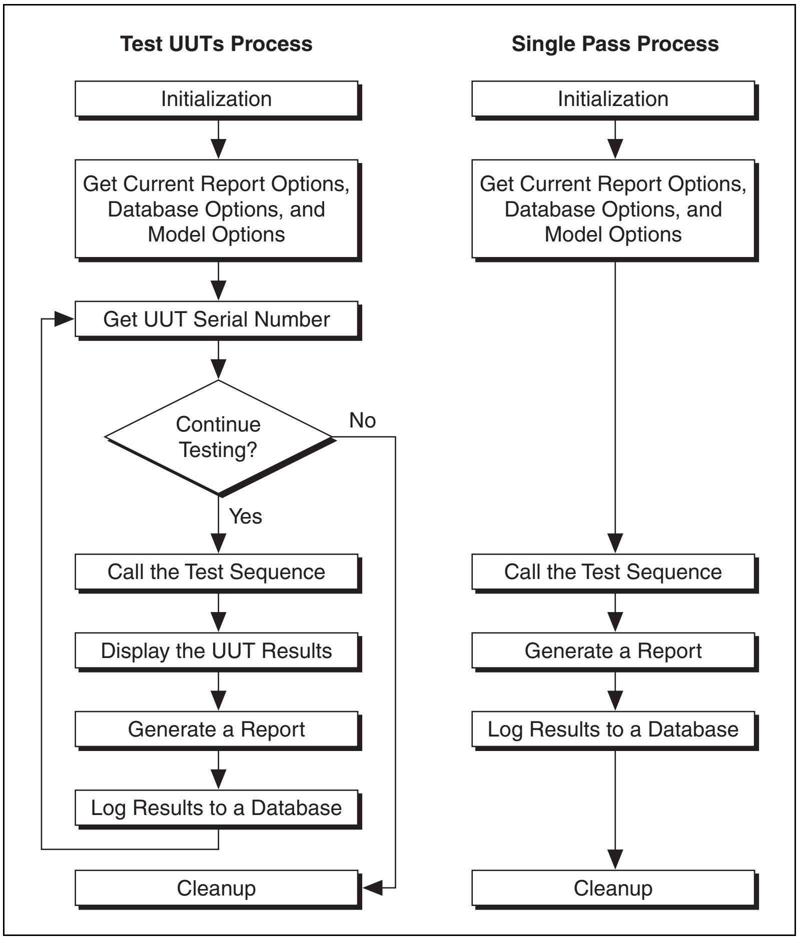


Figure 6-1. TestStand Sequential Model Callbacks

Viewing Process Model Callbacks

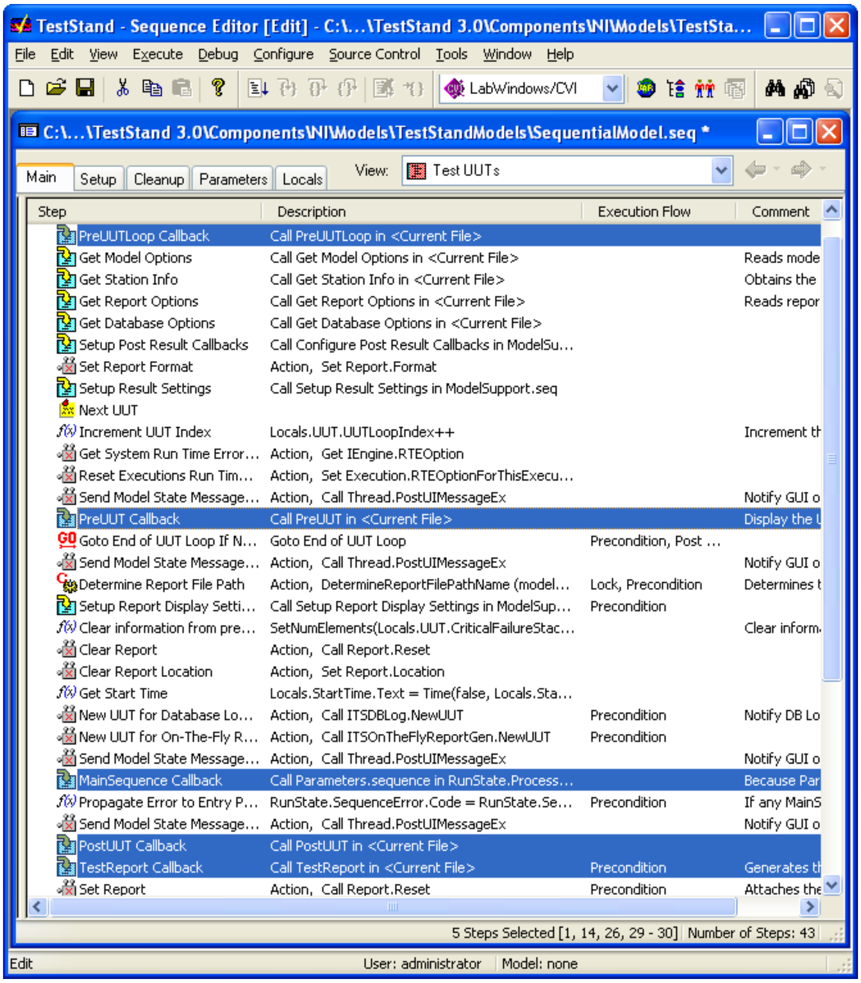
In this tutorial, you will examine the Model callbacks in the Sequential process model.

1. Open the file `<TestStand>\Components\NI\Models\TestStandModels\SequentialModel.seq`.

   This file, the Sequential model, is the default process model TestStand uses to execute sequences.

2. Select the Test UUTs sequence from the View ring control. This sequence is the Test UUTs Execution entry point that TestStand executes when you select **Execute»Test UUTs**.

   Notice the callback sequences that the Test UUTs sequence calls—PreUUTLoop Callback, PreUUT Callback, MainSequence Callback, PostUUT Callback, TestReport Callback, Log to Database Callback, and PostUUTLoop Callback—some of which are shown in Figure 6-2.


TestStand - Sequence Editor [Edit] - C:\...\TestStand 3.0\Components\NI\Models\TestSta...
File Edit View Execute Debug Configure Source Control Tools Window Help
LabWindows/CVI
C:\...\TestStand 3.0\Components\NI\Models\TestStandModels\SequentialModel.seq *
Main Setup Cleanup Parameters Locals
View: Test UUTs

Step	Description	Execution Flow	Comment
PreUUTLoop Callback	Call PreUUTLoop in <Current File>		
Get Model Options	Call Get Model Options in <Current File>		Reads mode
Get Station Info	Call Get Station Info in <Current File>		Obtains the
Get Report Options	Call Get Report Options in <Current File>		Reads repor
Get Database Options	Call Get Database Options in <Current File>		
Setup Post Result Callbacks	Call Configure Post Result Callbacks in ModelSu...		
Set Report Format	Action, Set Report.Format		
Setup Result Settings	Call Setup Result Settings in ModelSupport.seq		
Next UUT			
Increment UUT Index	Locals.UUT.UUTLoopIndex++		Increment th
Get System Run Time Error...	Action, Get IEngine.RTEOption		
Reset Executions Run Tim...	Action, Set Execution.RTEOptionForThisExecu...		
Send Model State Message...	Action, Call Thread.PostUIMessageEx		Notify GUI o
PreUUT Callback	Call PreUUT in <Current File>		Display the U
Goto End of UUT Loop If N...	Goto End of UUT Loop	Precondition, Post ...	
Send Model State Message...	Action, Call Thread.PostUIMessageEx		Notify GUI o
Determine Report File Path	Action, DetermineReportFilePathName (model...	Lock, Precondition	Determines t
Setup Report Display Setti...	Call Setup Report Display Settings in ModelSup...	Precondition	
Clear information from pre...	SetNumElements(Locals.UUT.CriticalFailureStac...		Clear inform.
Clear Report	Action, Call Report.Reset		
Clear Report Location	Action, Set Report.Location		
Get Start Time	Locals.StartTime.Text = Time(false, Locals.Sta...		
New UUT for Database Lo...	Action, Call ITSDBLog.NewUUT	Precondition	Notify DB Lo
New UUT for On-The-Fly R...	Action, Call ITSOnTheFlyReportGen.NewUUT	Precondition	
Send Model State Message...	Action, Call Thread.PostUIMessageEx		Notify GUI o
MainSequence Callback	Call Parameters.sequence in RunState.Process...		Because Par
Propagate Error to Entry P...	RunState.SequenceError.Code = RunState.Se...	Precondition	If any MainS
Send Model State Message...	Action, Call Thread.PostUIMessageEx		Notify GUI o
PostUUT Callback	Call PostUUT in <Current File>		
TestReport Callback	Call TestReport in <Current File>	Precondition	Generates th
Set Report	Action, Call Report.Reset	Precondition	Attaches the

5 Steps Selected [1, 14, 26, 29 - 30] Number of Steps: 43

Edit User: administrator Model: none

Figure 6-2. Test UUTs Sequence

3. Right-click the `PreUUT Callback` step and select **Open Sequence** from the context menu. The PreUUT Callback sequence has two steps: Identify UUT and Set Serial Number.
4. Right-click the `Identify UUT` step and select **Run Selected Steps** from the context menu. This launches the UUT Information dialog box, which is similar to the dialog box you see when you execute a sequence using the Test UUTs Execution entry point.

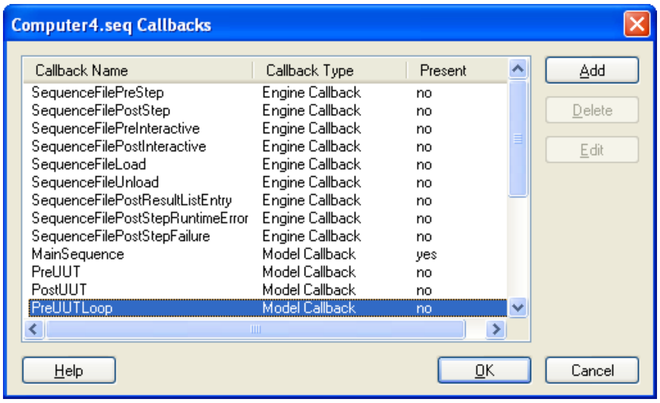
Tip To change the way in which TestStand obtains a UUT serial number, such as reading it from a bar code, replace this callback with your own custom callback.

5. Click **OK** in the UUT Information dialog box.
6. Close the Execution window.
7. Select **Test UUTs** again from the View ring control.
8. Right-click the `PreUUTLoop Callback` step and select **Open Sequence** from the context menu. Notice that this callback is empty. The empty sequence is a place holder. If you want to add steps that execute before the UUT loop, you can create them in this callback.
9. Close the `SequentialModel.seq` Sequence File window, and decline any prompts to save your changes.

Overriding a Process Model Callback

In the next tutorial, you will override the default `PreUUTLoop Callback` sequence in the Sequential model by creating a `PreUUTLoop Callback` sequence in a client sequence file.

1. Open the file `<TestStand>\Tutorial\Computer4.seq`, which you created in Chapter 5, *Using Variables and Properties*. You can also find this file in the `<TestStand>\Tutorial\Solution` directory.
2. Select **Edit»Sequence File Callbacks** to display the Sequence File Callbacks dialog box, shown in Figure 6-3.



Callback Name	Callback Type	Present
SequenceFilePreStep	Engine Callback	no
SequenceFilePostStep	Engine Callback	no
SequenceFilePreInteractive	Engine Callback	no
SequenceFilePostInteractive	Engine Callback	no
SequenceFileLoad	Engine Callback	no
SequenceFileUnload	Engine Callback	no
SequenceFilePostResultListEntry	Engine Callback	no
SequenceFilePostStepRuntimeError	Engine Callback	no
SequenceFilePostStepFailure	Engine Callback	no
MainSequence	Model Callback	yes
PreUUT	Model Callback	no
PostUUT	Model Callback	no
PreUUTLoop	Model Callback	no

Figure 6-3. Adding Callbacks to a Sequence

3. Select the `PreUUTLoop` callback.
4. Click **Add**.

   Notice that the value in the Present column changes from `no` to `yes`. When you click Add, the sequence editor creates a new empty callback sequence to your sequence file. Now, when you start an execution using an Execution entry point, TestStand calls the callback in your sequence file instead of the sequence in the Sequential model.
5. Click **OK** to close the Sequence File Callbacks dialog box.
6. Select **All Sequences** in the View ring control of the Sequence File window. Notice that the sequence file now contains two sequences: `MainSequence` and `PreUUTLoop`.
7. Right-click the `PreUUTLoop` sequence and select **View Contents** from the context menu.
8. Right-click the empty step list of the **Main** tab and select **Insert Step»Message Popup** from the context menu.
9. Rename the new step `Pre UUT Loop Message`.
10. Right-click the `Pre UUT Loop Message` step and select **Edit Message Settings** from the context menu.
    a. In the **Title Expression** control, enter the literal string `"Pre UUT Loop Callback Message"`.
    b. In the **Message Expression** control, enter the literal string `"Now in the Pre UUT Loop Callback"`.
    c. Click **OK** to close the Edit Message Settings dialog box.

11. Select **File»Save As**. Save the sequence as `Computer5.seq` in the `<TestStand>\Tutorial` directory.
12. Execute the sequence by selecting **Execute»Test UUTs**. Notice that the Pre UUT Loop Callback Message dialog box is the first prompt TestStand displays.
13. Click **OK** to close the Pre UUT Loop Callback Message dialog box. TestStand now displays the UUT Information dialog box from the PreUUT Callback sequence in the Sequential model.
14. Enter a serial number and click **OK**.
15. Run through several iterations of the sequence.
16. Click **Stop** in the UUT Information dialog box.

    Notice that TestStand only displays the Pre UUT Loop Callback Message dialog box once at the very beginning of the execution. This is because the PreUUTLoop Callback is executed before the loop, while the PreUUT Callback is executed within the loop, as shown in Figure 6-2.
17. Close all windows in the sequence editor.

You can make modifications similar to those in this tutorial to the other TestStand process model files, which are discussed in Chapter 2, *Loading and Running Sequences*.

For more information about the process models, callbacks, and modifying callbacks, refer to Chapter 1, *TestStand Architecture*, Chapter 10, *Customizing Process Models and Callbacks*, and Appendix A, *Process Model Architecture*, in the *TestStand Reference Manual*.

You have completed this tutorial. In the next chapter, you will learn how to add users and set or modify user privileges.

7

Adding Users and Setting Privileges

The TestStand Sequence Editor includes a user manager for adding and removing users, and for managing the privileges of each user. This chapter discusses how to use the TestStand User Manager, how to add new users, and how to change user privileges.

Using the User Manager

Note The TestStand User Manager is designed to help you implement policies and procedures concerning the use of your test station. It is not a security system and it does not inhibit or control the operating system or third-party applications. You must use the system level security features provided by your operating system to secure your test station computer against malicious use.

Adding a New User

In this tutorial, you will learn how to add a new user.

Note This tutorial assumes that you are currently logged in as the `administrator` user. If you are not logged in as `administrator`, select **File»Login** and log in as `administrator`. Leave the password field empty.

1. Launch the TestStand User Manager window by selecting **View»User Manager**. You can also click the **User Manager** button on the toolbar.

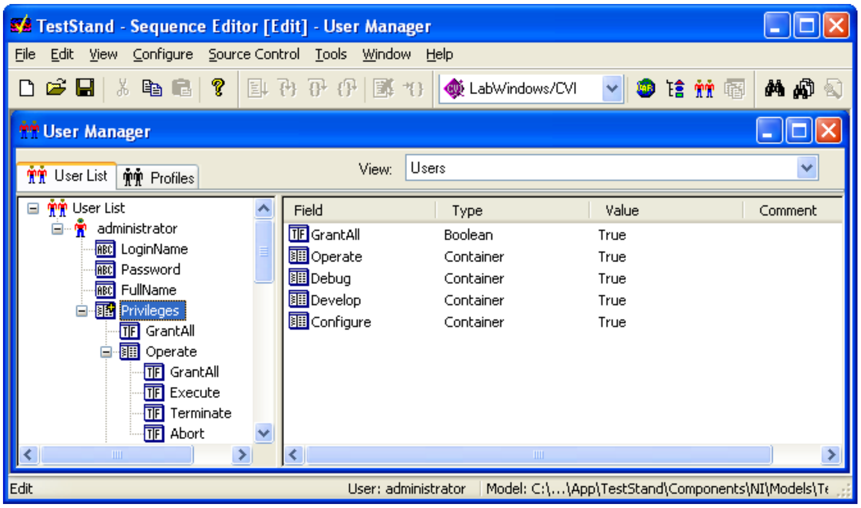
   The left pane of the user manager window shows a tree view of all of the users configured on your test station.

2. Expand the `administrator` node to display the hierarchy of properties associated with the *administrator* user profile.

   The `Privileges` node contains settings for all of the actions a user can perform, such as executing sequences, debugging sequences, or adding new users.

   When you select a node in the tree view, the corresponding property values under the node appear in the list pane on the right. Notice that all of the property values for the privileges under the `administrator` user are `True`, as shown in Figure 7-1.

You will learn more about privileges and how to modify them in the *Modifying User Privileges* section of this tutorial.


TestStand - Sequence Editor [Edit] - User Manager
File Edit View Configure Source Control Tools Window Help
LabWindows/CVI
User Manager
User List
Profiles
View: Users

```
User List
  administrator
    LoginName
    Password
    FullName
    Privileges
      GrantAll
      Operate
        GrantAll
        Execute
        Terminate
        Abort
```

Field	Type	Value	Comment
GrantAll	Boolean	True	
Operate	Container	True	
Debug	Container	True	
Develop	Container	True	
Configure	Container	True	

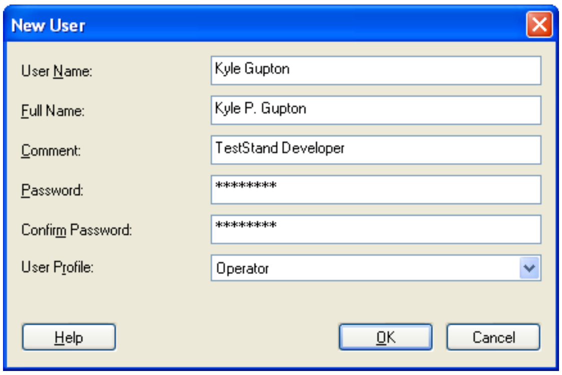
```
Edit                User: administrator | Model: C:\...\App\TestStand\Components\NI\Models\Te
```

Figure 7-1. User Manager Window

3. Click the `User List` node in the tree view, which lists the `administrator` user in the right pane.
4. Right-click in the right pane and select **Insert User** from the context menu to launch the New User dialog box. Complete the New User dialog box using the following steps:
    a. Enter your name in the **User Name** and **Full Name** controls.
    b. Type a password into the **Password** and **Confirm Password** controls.
    c. For the **User Profile** control, select `Operator`.

    User profiles define an initial set of privilege settings for the new user. TestStand provides four default user profiles: `Operator`, `Technician`, `Developer`, and `Administrator`. By default, the `Operator` profile grants a user the privilege to execute, *terminate*, and *abort* sequences, but does not grant the privilege to create or debug sequences.

    Figure 7-2 shows an example of the completed New User dialog box.



New User

User Name: Kyle Gupton

Full Name: Kyle P. Gupton

Comment: TestStand Developer

Password: ********

Confirm Password: ********

User Profile: Operator

Figure 7-2. New User Dialog Box

   d. Click **OK** to close the New User dialog box.

5. Select **File»Login**. The new user you just added should now appear in addition to the `administrator` user name in the User Name ring control of the Login dialog box.
6. Log in as `administrator`.

Creating a New User Profile

The user manager also allows you to modify the default profiles and to create new profiles that define a combination of privileges appropriate for your test station.

Complete the following steps to create a new user profile:

1. Click the **Profiles** tab of the User Manager window. Notice the four default profiles in the list pane.
2. Click the `Profiles` node in the tree view. This lists the currently defined profiles in the right list pane.
3. Right-click the `Operator` profile in the right pane and select **Copy** from the context menu.
4. Click the `Profiles` node again. Right-click in the right pane and select **Paste**.
5. Rename the new profile `Senior Operator`. The new profile is now identical to the `Operator` profile.

Note If you make changes to the values in a profile, your changes do not affect the privileges for users that already exist in the user list. After you create a user, you must modify privileges individually. You cannot modify privileges for existing users by changing user profiles.

Modifying User Privileges

Privileges are organized in hierarchical groups. Each privilege group has a Boolean subproperty named GrantAll. A user has a privilege if you set the property of the privilege to `True`. Alternatively, you can set the GrantAll property of a privilege group to specify whether a user has all privileges within a privilege group, regardless of the property value of the individual privileges.

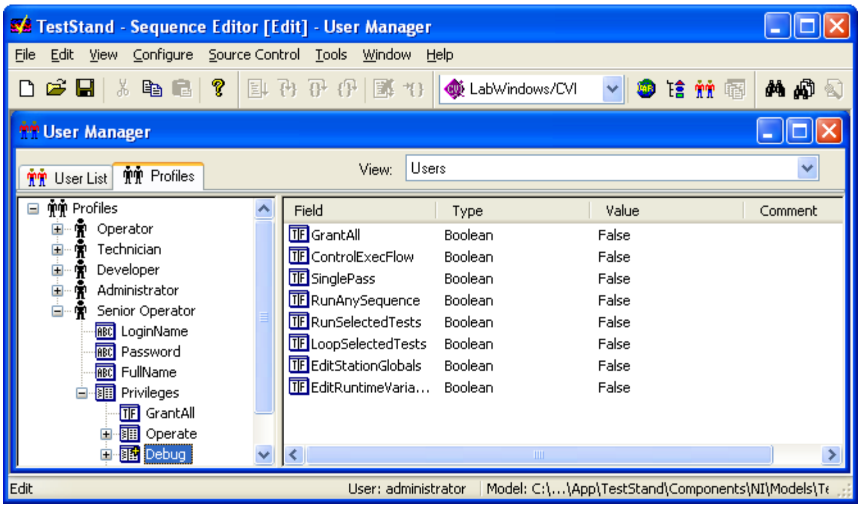
Note The property User.Privileges.GrantAll applies to all privilege groups. If this property is set to `True`, the user has all privileges. This property must be set to `False` to honor privilege settings within each privilege group.

You can grant privileges in several different ways. For example, a user has the privilege to terminate an execution if one of the following set of conditions is met:

- User.Privileges.GrantAll is set to `True`. This also grants rights to all other privileges.
- User.Privileges.GrantAll is set to `False` and User.Privileges.Operate.GrantAll is set to `True`. This also grants rights to other privileges of the Operate privilege group.
- User.Privileges.GrantAll and User.Privileges.Operate.GrantAll are set to `False` and User.Privilege.Operate.Terminate is set to `True`. This only ensures that a user has the privilege to terminate an execution.

To modify the default privileges for the profile you created in the *Creating a New User Profile* section of this tutorial, complete the following steps:

1. Select the `Senior Operator` node in the tree view and expand its privilege settings.
2. Select the `Debug` node in the tree view, as shown in Figure 7-3.



View: Users

Field	Type	Value	Comment
GrantAll	Boolean	False	
ControlExecFlow	Boolean	False	
SinglePass	Boolean	False	
RunAnySequence	Boolean	False	
RunSelectedTests	Boolean	False	
LoopSelectedTests	Boolean	False	
EditStationGlobals	Boolean	False	
EditRuntimeVaria...	Boolean	False	

Edit User: administrator Model: C:\...\App\TestStand\Components\NI\Models\Te

Figure 7-3. Configure Privileges in New Profile

The `Debug` node is a *Container property*, which contains Boolean subproperties. The values of the subproperties under the `Debug` node are `False`.

You can set the value of each privilege in the right pane by right-clicking an item and selecting **Properties** from the context menu.

3. Set the `SinglePass` property under the `Debug` node to `True` for the `Senior Operator` profile, as follows:

   a. Double-click `Single Pass` in the right pane.

      When you make this selection, TestStand launches the Boolean Properties dialog box. The title of the Boolean Properties dialog box is specific to the step that you select. In this case, the Boolean Properties dialog box is named SinglePass Properties.

   b. Change the value to `True`.

   c. Click **OK** to close the SinglePass Properties dialog box.

4. Add a new user using the `Senior Operator` profile, as follows:

   a. Click the **User List** tab.

   b. Right-click in the right pane and select **Insert User** from the context menu.

   c. Enter the information in the New User dialog box as you did in step 4 of the *Adding a New User* section of this tutorial. Complete the New User dialog box using a different user name.

   d. Select the `Senior Operator` profile.
   e. Click **OK** to close the New User dialog box.

5. Select **File»Login**. The new user you just added should now appear in addition to the `administrator` user name in the User Name ring control of the Login dialog box.
6. Select the user you created with the `Senior Operator` profile in step 4. Do not specify the user that you created in the *Adding a New User* section of this tutorial.
7. Enter the appropriate password.
8. Click **OK**.
9. Open `Computer.seq` from the `<TestStand>\Tutorial` directory.
10. Select **Execute»Single Pass**.

    Notice that you can select Single Pass from the Execute menu, but the Run MainSequence option is grayed out. This is because you no longer have the privilege to execute sequences without a Model entry point.
11. Try to right-click in a sequence view to insert a new step. Notice that the insert menu command is also grayed out because your privileges have changed.
12. Close all of the windows in the sequence editor.
13. Select **File»Login**.
14. Log in as `administrator`.

You can also add and remove users programatically using the TestStand API. The shipping example, `<TestStand>\Example\CreateDeleteUsers\CreateDeleteUsers.seq`, demonstrates how to add and remove users using the TestStand API.

You have completed this tutorial. In the next chapter, you will learn how to run steps in interactive mode.

8 Interactive Executions

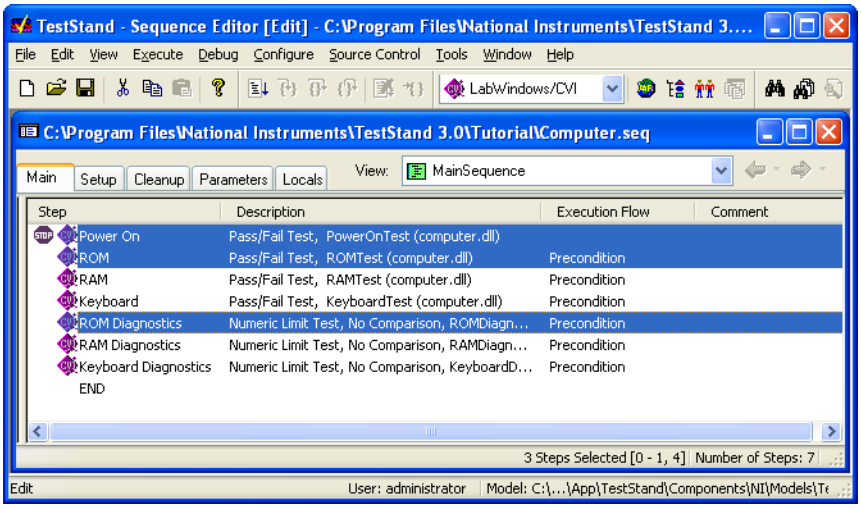
In this chapter, you learn how to run steps in *interactive mode*. In interactive mode, you can execute specific steps in a sequence.

Running Selected Steps as a Separate Execution

In this tutorial, you run selected steps from a Sequence File window as a separate execution.

1. Open `Computer.seq` from the `<TestStand>\Tutorial` directory.
2. Insert a breakpoint at the `Power On` step by clicking to the left of the step icon, or by right-clicking the step and selecting **Breakpoint»Toggle Breakpoint** from the context menu.
3. Select the `Power On`, `ROM`, and `ROM Diagnostics` steps by holding down the **<Ctrl>** key and clicking each step.

   The resulting Sequence File window is shown in Figure 8-1.


TestStand - Sequence Editor [Edit] - C:\Program Files\National Instruments\TestStand 3....
File Edit View Execute Debug Configure Source Control Tools Window Help
LabWindows/CVI
C:\Program Files\National Instruments\TestStand 3.0\Tutorial\Computer.seq
Main Setup Cleanup Parameters Locals
View: MainSequence

Step	Description	Execution Flow	Comment
Power On	Pass/Fail Test, PowerOnTest (computer.dll)		
ROM	Pass/Fail Test, ROMTest (computer.dll)	Precondition	
RAM	Pass/Fail Test, RAMTest (computer.dll)	Precondition	
Keyboard	Pass/Fail Test, KeyboardTest (computer.dll)	Precondition	
ROM Diagnostics	Numeric Limit Test, No Comparison, ROMDiagn...	Precondition	
RAM Diagnostics	Numeric Limit Test, No Comparison, RAMDiagn...	Precondition	
Keyboard Diagnostics	Numeric Limit Test, No Comparison, KeyboardD...	Precondition	
END			

3 Steps Selected [0 - 1, 4] Number of Steps: 7

Edit User: administrator Model: C:\...\App\TestStand\Components\NI\Models\Te

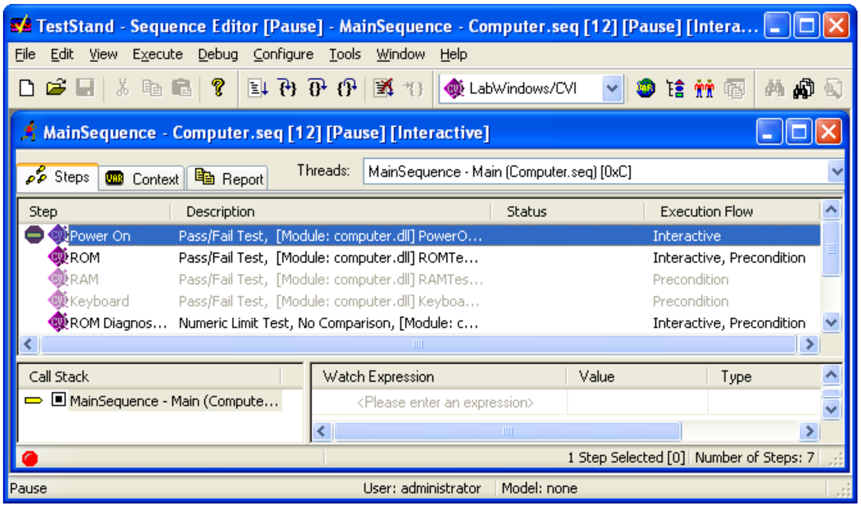
Figure 8-1. Selecting Multiple Steps in a Sequence File Window

4. Select **Execute»Run Selected Steps**. This command launches the Test Simulator dialog box and starts a new execution.

   The new execution is called a *root interactive execution*. By default, when you run selected steps from a Sequence File window, TestStand executes the Setup, Main, and Cleanup step groups. You can modify these settings from the Station Options dialog box to control whether the Setup and Cleanup step groups run as part of the root interactive execution.

5. Click **Done** to close the Test Simulator dialog box.

6. The execution stops at the `Power On` step breakpoint, as shown in Figure 8-2.


TestStand - Sequence Editor [Pause] - MainSequence - Computer.seq [12] [Pause] [Intera...
File Edit View Execute Debug Configure Tools Window Help
LabWindows/CVI
MainSequence - Computer.seq [12] [Pause] [Interactive]
Steps Context Report
Threads: MainSequence - Main (Computer.seq) [0xC]

Step	Description	Status	Execution Flow
Power On	Pass/Fail Test, [Module: computer.dll] PowerO...		Interactive
ROM	Pass/Fail Test, [Module: computer.dll] ROMTe...		Interactive, Precondition
RAM	Pass/Fail Test, [Module: computer.dll] RAMTes...		Precondition
Keyboard	Pass/Fail Test, [Module: computer.dll] Keyboa...		Precondition
ROM Diagnos...	Numeric Limit Test, No Comparison, [Module: c...		Interactive, Precondition

Call Stack
MainSequence - Main (Compute...

Watch Expression	Value	Type
<Please enter an expression>		

1 Step Selected [0] Number of Steps: 7

Pause User: administrator Model: none

Figure 8-2. Breakpoint During Interactive Execution

Notice that the pointer for the interactive execution is a narrow yellow arrow. The sequence editor uses a full yellow arrow for a *normal execution*.

7. Single-step through the execution by selecting **Debug»Step Over** twice. Notice that the execution executes only the steps that you had previously selected and that the non-selected steps are dimmed.

8. Complete the execution by selecting **Debug»Resume**. Notice that TestStand ignores the preconditions for the `ROM Diagnostics` step and runs the step even though the `ROM` step passed.

9. Close the Execution window after the execution completes.

10. Repeat steps 3 through 9, but select **Execute»Run Selected Steps Using»Single Pass** in step 4. TestStand executes your steps using the Single Pass Execution entry point, which produces a UUT report.

Running Selected Steps During an Execution

To interactively execute selected steps in a sequence while suspended at a breakpoint during an execution, complete the following steps:

1. Select **Execute»Single Pass** to start a new execution.
2. When the Test Simulator dialog box appears, select the ROM test to fail and click **Done**.

   The execution stops at the breakpoint on the `Power On` step.
3. Using **Debug»Step Over**, step through the execution until you reach the `RAM Diagnostics` step. Notice that the `ROM` step failed.
4. Place a second breakpoint at the `ROM` step in the Execution window.
5. Select the `ROM` and `ROM Diagnostics` steps by holding down the <Ctrl> key and clicking each step.
6. Right-click the `ROM Diagnostics` step and select **Loop on Selected Steps** from the context menu.
7. In the Loop on Selected Steps dialog box, enter the number `100` in the **Loop Count** control.
8. Click **OK** to close the Loop on Selected Steps dialog box.

   The sequence editor starts an interactive execution for the selected steps and enters a paused state at the breakpoint for the ROM step. Notice that the execution pointer for the *normal execution* is still on the RAM Diagnostics step and an execution pointer for the new interactive execution is now pointing to the ROM step.
9. Single-step through the interactive execution by selecting **Debug»Step Over**. Notice that the interactive execution only toggles between the ROM and the ROM Diagnostics steps.
10. Notice how the status of the `ROM` step does not pass, but continues to fail. Rather than complete 100 loops of the interactive execution, select **Debug»Terminate Interactive Execution**. When you make this selection, TestStand returns the execution to a paused state on the RAM Diagnostics step.

11. Complete the following steps to force the execution to continue from a step other than the currently paused step:
    a. Click the `ROM` step so that it is the only highlighted step.
    b. Right-click the `ROM` step and select **Set Next Step** from the context menu. Notice that the execution pointer moves from the RAM Diagnostics step to the ROM step.
    c. Select **Debug»Step Over to** single-step once and notice that the execution executes the ROM step instead of the RAM Diagnostics step.
12. Complete the execution by selecting **Debug»Resume**.

    When the report file completes, notice that the report contains entries for each interactively executed step.
13. Close all windows in the sequence editor and do not save any changes to the sequence file.

You have completed this tutorial. In the next chapter, you will learn how to dynamically call sequences and pass parameters.

9 Calling Sequences Dynamically

In this chapter, you learn how to add a step to a sequence that dynamically runs one of two sequences. You will also learn how to pass parameters to the sequence you call.

Dynamically Specifying a Sequence to Run

In this tutorial, you will open an existing sequence, add steps to prompt the operator for a CPU type and a number of CPUs to test, and add a step to call one of two different sequences, depending on the type of CPU the user specifies.

1. Open `Computer.seq` from the `<TestStand>\Tutorial` directory.
2. Click the **Locals** tab of the Sequence File window.
3. Right-click in the right pane and select **Insert Local»String** from the context menu.
4. Rename the local variable `CPUType`.
5. Click the **Main** tab in the Sequence File window to display the steps in the Main step group.
6. Right-click the `Power On` step and select **Insert Step»Message Popup** from the context menu.
7. Rename the new step `Select CPU Type`.
8. Right-click the `Select CPU Type` step and select **Edit Message Settings** from the context menu.
9. Under the **Text And Buttons** tab, change the following control values on the Configure Message Popup Step dialog box:

Title Expression	`"Select CPU"`
Message Expression	`"Please select the CPU Type for the UUT."`
Button 1	`"INTEL CPU"`
Button 2	`"AMD CPU"`
Cancel Button	`None`

10. Select the **Options** tab and enable the **Make Modal** option.

    Enabling this option prevents the Sequence Editor window from hiding the Message Popup dialog box and prevents you from interacting with the sequence editor until you close the Message Popup dialog box.
11. Click **OK** to close the Configure Message Popup Step dialog box.
12. Right-click the `Select CPU Type` step and select **Properties** from the context menu.
13. Click the **Expressions** tab.
14. Enter the following expression in the **Post Expression** control:

    ```
    Locals.CPUType = ((Step.Result.ButtonHit == 2) ?
    "AMD" : "INTEL")
    ```

    This expression assigns the string value `"AMD"` or `"INTEL"` to the local variable, depending on which button you click.

Tip You can also use the Expression Browser to create this expression. Click the **Expression Browse** button, located next to the Post Expression control, to launch the Expression Browser. You can also get descriptions of condition operators, such as `?` or `:`, in the Expression Browser and in the *TestStand Help*.

15. Click **OK** to close the Step Properties dialog box.
16. Right-click the `Select CPU Type` step and select **Insert Step»Message Popup** from the context menu.
17. Rename the new step `Specify Number of CPUs`.
18. Right-click the new step and select **Edit Message Settings** from the context menu.
19. On the **Text and Buttons** tab of the Configure Message Popup Step dialog box, change the following control values:

Title Expression	`"Number of CPUs"`
Message Expression	`"Please select the number of CPUs installed for the UUT."`
Button 1	`"1"`
Button 2	`"2"`
Button 3	`"3"`
Button 4	`"4"`
Cancel Button	`None`

20. Select the **Options** tab and enable the **Make Modal** option.

21. Click **OK** to close the Configure Message Popup Step dialog box.
22. Right-click the `Specify Number of CPUs` step and select **Insert Step»Sequence Call** in the context menu.
23. Rename the step `CPU Test`.
24. Right-click the `CPU Test` step and select **Specify Module** from the context menu.
25. Enable the **Specify Expressions for Pathname and Sequence** option.
26. Enter the following values for the **File Path or Reference** and **Sequence** controls:

File Path or Reference	`Locals.CPUType + "Processor.seq"`
Sequence	`"MainSequence"`

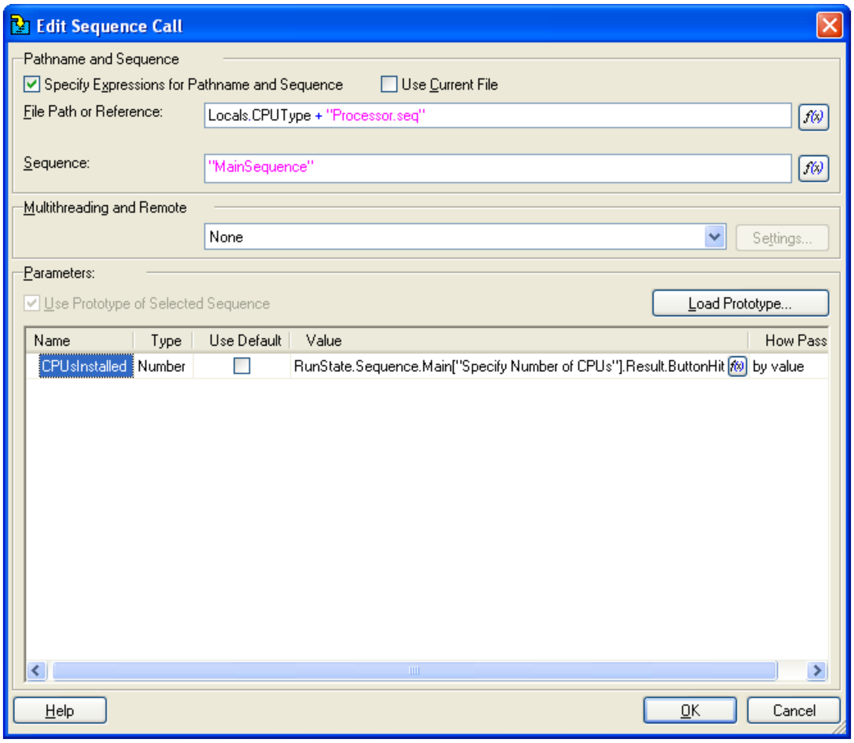
27. Select the prototype for the Sequence Call step by clicking the **Load Prototype** button.
28. Click the **File Browse** button in the Load Sequence Prototype dialog box.
29. Select the `AMDProcessor.seq` sequence file.
30. Click **OK** twice to close both the Select Sequence File and the Load Sequence Prototype dialog boxes.

    Notice that TestStand populates the Parameters section with the parameter list for the sequence.
31. Click in the **Value** column of the **CPUsInstalled** parameter.
32. Enter the following expression into the **Value** column field:

```
RunState.Sequence.Main["Specify Number of
CPUs"].Result.ButtonHit
```

Tip You can also locate the property through the Expression Browser dialog box, which you launch by clicking the **Expression Browse** button.

Figure 9-1 shows the completed Edit Sequence Call dialog box.



- [✓] Specify Expressions for Pathname and Sequence
- [ ] Use Current File

File Path or Reference: Locals.CPUType + "Processor.seq"

Sequence: "MainSequence"

Multithreading and Remote: None

Name	Type	Use Default	Value	How Passed
CPUsInstalled	Number	☐	RunState.Sequence.Main["Specify Number of CPUs"].Result.ButtonHit	by value

Figure 9-1. Dynamically Calling with an Expression

33. Click **OK** to close the Edit Sequence Call dialog box.
34. Select **File»Save As** and save the sequence in the `<TestStand>\Tutorial` directory as `Computer6.seq`.

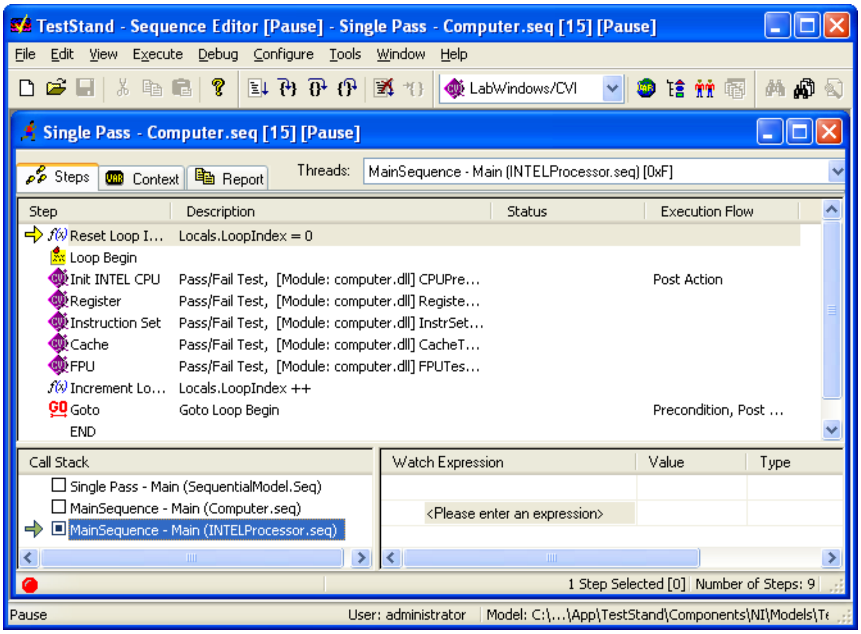
Running a Sequence Dynamically

Complete the following steps to run a sequence dynamically:

1. Place a breakpoint on the `CPU Test` step.
2. Select **Execute»Single Pass**.
3. Click **Done** in the Test Simulator dialog box.
4. Click **INTEL CPU** in the Select CPU dialog box.
5. Select **2** in the Number of CPUs dialog box.

6. After the execution pauses at the breakpoint on the `CPU Test` step, single-step into the subsequence by selecting **Debug»Step Into**.

   Notice that the Call Stack pane lists `INTELProcessor.seq` at the bottom of the sequence call stack, as shown in Figure 9-2.


TestStand - Sequence Editor [Pause] - Single Pass - Computer.seq [15] [Pause]
File Edit View Execute Debug Configure Tools Window Help
LabWindows/CVI
Single Pass - Computer.seq [15] [Pause]
Steps Context Report
Call Stack
Pause

Threads: MainSequence - Main (INTELProcessor.seq) [0xF]

Step	Description	Status	Execution Flow
Reset Loop I...	Locals.LoopIndex = 0		
Loop Begin			
Init INTEL CPU	Pass/Fail Test, [Module: computer.dll] CPUPre...		Post Action
Register	Pass/Fail Test, [Module: computer.dll] Registe...		
Instruction Set	Pass/Fail Test, [Module: computer.dll] InstrSet...		
Cache	Pass/Fail Test, [Module: computer.dll] CacheT...		
FPU	Pass/Fail Test, [Module: computer.dll] FPUTes...		
Increment Lo...	Locals.LoopIndex ++		
Goto	Goto Loop Begin		Precondition, Post ...
END			

Call Stack:
- Single Pass - Main (SequentialModel.Seq)
- MainSequence - Main (Computer.seq)
- MainSequence - Main (INTELProcessor.seq)

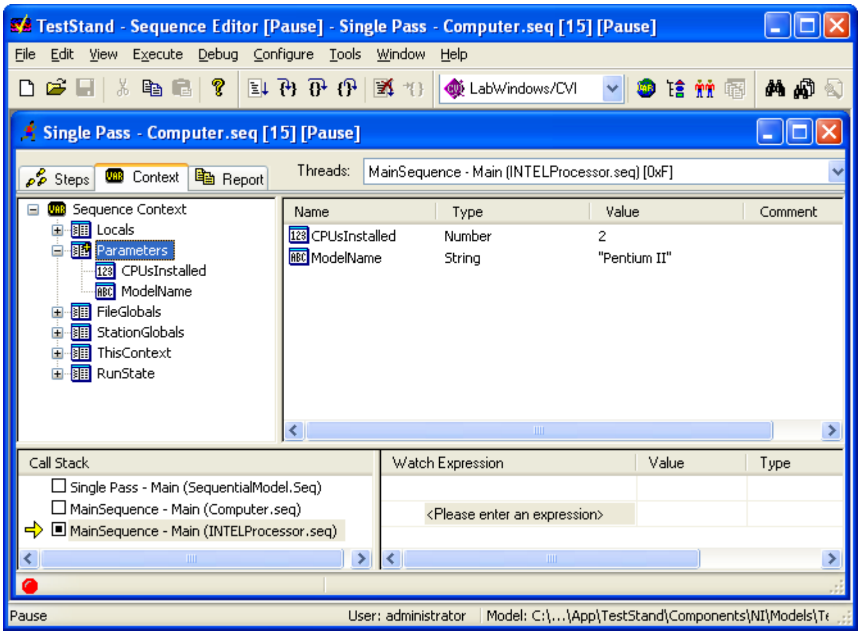
Watch Expression	Value	Type
<Please enter an expression>		

1 Step Selected [0] Number of Steps: 9

User: administrator Model: C:\...\App\TestStand\Components\NI\Models\Te

Figure 9-2. INTELProcessor.seq in the Call Stack Pane

7. Click the **Context** tab. Notice the values of the two parameters for the sequence, which are shown in Figure 9-3.


TestStand - Sequence Editor [Pause] - Single Pass - Computer.seq [15] [Pause]
File Edit View Execute Debug Configure Tools Window Help
LabWindows/CVI
Single Pass - Computer.seq [15] [Pause]
Steps
Context
Report
Threads:
Sequence Context
Locals
Parameters
CPUsInstalled
ModelName
FileGlobals
StationGlobals
ThisContext
RunState
Pause

Threads: MainSequence - Main (INTELProcessor.seq) [0xF]

Name	Type	Value	Comment
CPUsInstalled	Number	2	
ModelName	String	"Pentium II"	

Call Stack
- Single Pass - Main (SequentialModel.Seq)
- MainSequence - Main (Computer.seq)
- MainSequence - Main (INTELProcessor.seq)

Watch Expression	Value	Type
<Please enter an expression>		

User: administrator Model: C:\...\App\TestStand\Components\NI\Models\Te

Figure 9-3. Sequence Parameters in the Context Tab

The value of the CPUsInstalled parameter is equal to the value on the button you clicked in the Specify Number of CPUs dialog box. Notice that `MainSequence` in the `INTELProcessor.seq` sequence file also requires a ModelName parameter. The Sequence Call step you created did not specify this parameter, so the engine initializes the parameter value to its default value.

8. Complete the execution by selecting **Debug»Resume**.
9. When the execution completes, review the report, but do not close the Execution window.
10. Restart the execution by selecting **Execute»Restart**.
11. Click **Done** in the Test Simulator dialog box.
12. Click the **AMD CPU** button in the Select CPU dialog box.
13. Select **3** in the Number of CPUs dialog box.

14. When the execution pauses at the breakpoint on the `CPU Test` step, step into the subsequence by selecting **Debug»Step Into**.

    Notice that the Call Stack pane lists `AMDProcessor.seq` at the bottom of the call stack.
15. Complete the execution by selecting **Debug»Resume**, and review the report.
16. Close the Execution and Sequence File windows.

You have completed this tutorial. In the next chapter, you will learn how to customize the reports that TestStand generates.

10 Customizing the Report

In this chapter, you will learn how to customize report generation within TestStand. Through the callback structure examined in Chapter 6, *Using Callbacks*, you can create your own TestReport Callback routine to develop reports in any format. Because changing report generation is so common, TestStand provides several options to configure the format of the test report without creating your own callback.

Configuring Test Report Options

1. Open `Computer.seq` from the `<TestStand>\Tutorial` directory.
2. Select **Configure»Report Options**. Follow the steps below to configure the Report Options dialog box:
   a. Enable the **Include Step Results** option and configure the following step result settings:
      – Set the **Result Filtering Expression** control to **Exclude/Passed/Done/Skipped** by clicking the arrow to the right of the control.

        The Result Filtering Expression control determines the conditions that must be met before the results are logged to the test report. In this example, you configure TestStand to record only the results of the steps that do not pass or steps that complete without any status. TestStand evaluates the expression at run time when generating the report. The results of each step are only logged to the test report if the expression is `True`.

      – Enable the **Include Test Limits** option.

      – Enable the **Include Measurements** option.

        In this example, when a measurement is an array, you configure TestStand to include the array as a table by selecting the **Insert Table** option under the Include Arrays control. You can also include it as a graph if you are producing HTML- or XML-based reports.

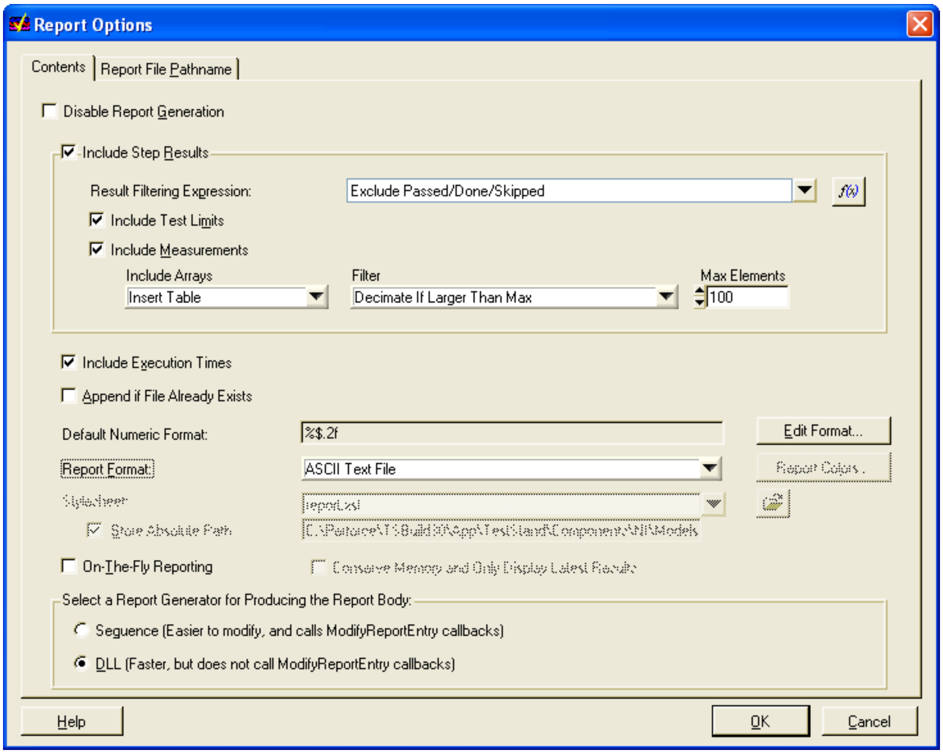
   b. Enable the **Include Execution Times** option.

c. Click **Edit Format** to display the Numeric Format dialog box, which allows you to configure how TestStand stores numeric values in the test report. By default, TestStand configures the numeric format to report numbers with 13 digits of precision.

d. Change **Number of Fractional Digits** to **2** and click **OK to** close the Edit Numeric Format dialog box.

e. Set the **Report Format** control to **ASCII Text File**. This setting creates the test report in a standard ASCII format.

The completed Report Options dialog box is shown in Figure 10-1.



☐ Disable Report Generation

☑ Include Step Results

Result Filtering Expression: Exclude Passed/Done/Skipped

☑ Include Test Limits

☑ Include Measurements

Include Arrays: Insert Table

Filter: Decimate If Larger Than Max

Max Elements: 100

☑ Include Execution Times

☐ Append if File Already Exists

Default Numeric Format: %$.2f

Report Format: ASCII Text File

Stylesheet: report.xsl

☑ Store Absolute Path: C:\Perforce\TSBuild30\App\TestStand\Components\NI\Models

☐ On-The-Fly Reporting

☐ Conserve Memory and Only Display Latest Results

Select a Report Generator for Producing the Report Body:

○ Sequence (Easier to modify, and calls ModifyReportEntry callbacks)

● DLL (Faster, but does not call ModifyReportEntry callbacks)

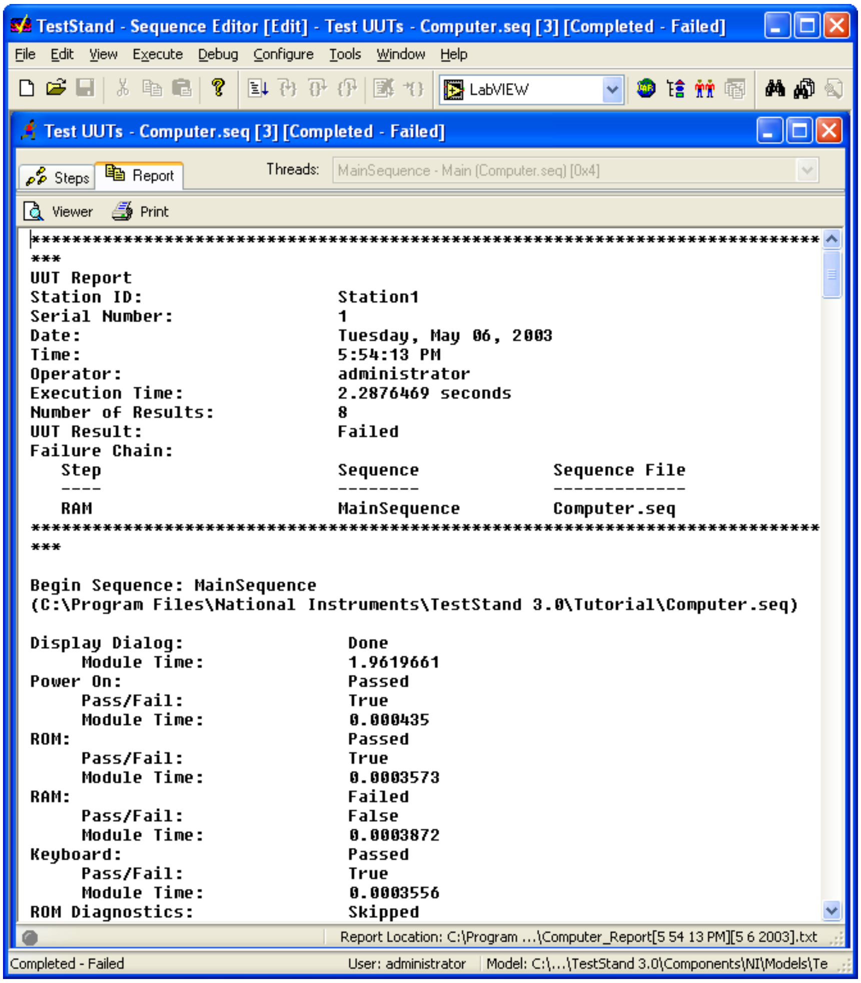
Figure 10-1. Report Options Dialog Box

3. Click the **Report File Pathname** tab.

Use this tab to configure the name and path for the test report file. You can, for example, create a new file for each UUT, or include the time and date in the name of the report file. Leave the tab options set to their default values and click **OK** to close the dialog box.

4. Execute the sequence by selecting **Execute»Test UUTs**.
5. Run through several iterations of the sequence and select different components, other than the Video and CPU tests, to fail.
6. Click **Stop** in the UUT Information dialog box to stop sequence execution.

   Notice that the test report contains failure chain information for UUTs that fail. The failure chain shows the step whose failure caused the UUT to fail, as well as the Sequence Call steps through which the execution reaches the failing step. Figure 10-2 shows a failure chain in which the failure of the RAM step in `Computer.seq` causes the UUT whose Serial Number is 1 to fail. Also, notice that the only step results that appear in the report are for steps that failed. The report format should resemble Figure 10-2.


TestStand - Sequence Editor [Edit] - Test UUTs - Computer.seq [3] [Completed - Failed]
File Edit View Execute Debug Configure Tools Window Help
LabVIEW
Test UUTs - Computer.seq [3] [Completed - Failed]
Steps
Report
Threads: MainSequence - Main (Computer.seq) [0x4]
Viewer
Print

```
******************************************************************************
***
UUT Report
Station ID:                   Station1
Serial Number:                1
Date:                         Tuesday, May 06, 2003
Time:                         5:54:13 PM
Operator:                     administrator
Execution Time:               2.2876469 seconds
Number of Results:            8
UUT Result:                   Failed
Failure Chain:
   Step                       Sequence           Sequence File
   ----                       --------           -------------
   RAM                        MainSequence       Computer.seq
******************************************************************************
***

Begin Sequence: MainSequence
(C:\Program Files\National Instruments\TestStand 3.0\Tutorial\Computer.seq)

Display Dialog:                Done
     Module Time:              1.9619661
Power On:                      Passed
     Pass/Fail:                True
     Module Time:              0.000435
ROM:                           Passed
     Pass/Fail:                True
     Module Time:              0.0003573
RAM:                           Failed
     Pass/Fail:                False
     Module Time:              0.0003872
Keyboard:                      Passed
     Pass/Fail:                True
     Module Time:              0.0003556
ROM Diagnostics:               Skipped
```

Report Location: C:\Program ...\Computer_Report[5 54 13 PM][5 6 2003].txt

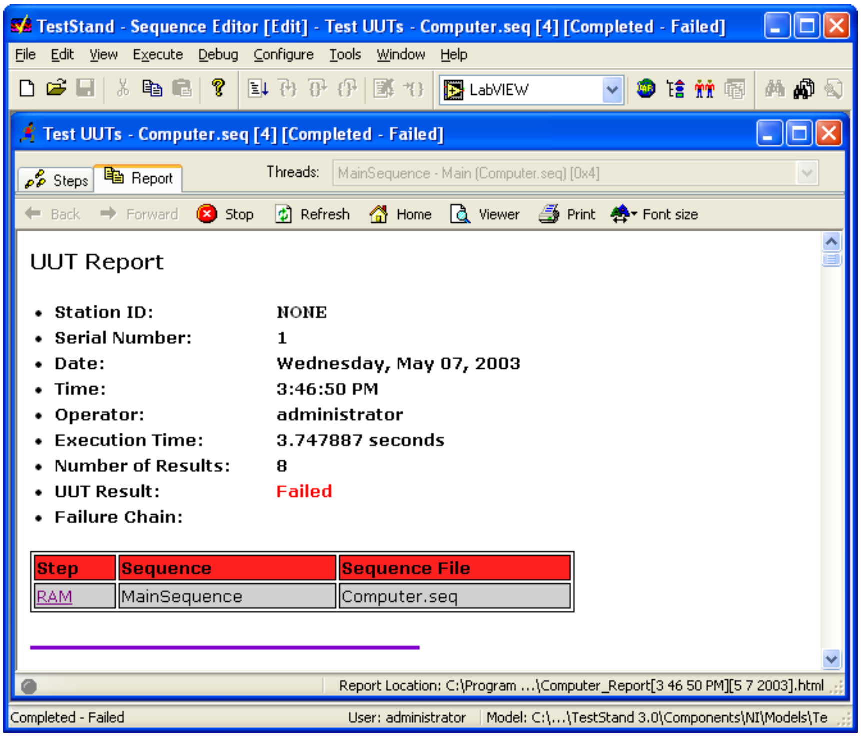
Completed - Failed   User: administrator   Model: C:\...\TestStand 3.0\Components\NI\Models\Te

Figure 10-2. Test Report in Text Format

7. Close the Execution window.
8. Select **Configure»Report Options**.
9. Select **HTML Document** or **XML Document** in the Report Format ring control.
10. Click **OK** to close the Report Options dialog box.
11. Repeat steps 4 through 6.

12. Examine the test report.

    Notice that in the HTML and XML reports, each step name in the failure chain is a hyperlink to the section of the report that displays the result for the step. The report format should resemble Figure 10-3.


TestStand - Sequence Editor [Edit] - Test UUTs - Computer.seq [4] [Completed - Failed]
File Edit View Execute Debug Configure Tools Window Help
LabVIEW
Test UUTs - Computer.seq [4] [Completed - Failed]
Steps Report
Threads: MainSequence - Main (Computer.seq) [0x4]
Back Forward Stop Refresh Home Viewer Print Font size

UUT Report

- Station ID: NONE
- Serial Number: 1
- Date: Wednesday, May 07, 2003
- Time: 3:46:50 PM
- Operator: administrator
- Execution Time: 3.747887 seconds
- Number of Results: 8
- UUT Result: Failed
- Failure Chain:

Step	Sequence	Sequence File
RAM	MainSequence	Computer.seq

Report Location: C:\Program ...\Computer_Report[3 46 50 PM][5 7 2003].html

Completed - Failed User: administrator Model: C:\...\TestStand 3.0\Components\NI\Models\Te

Figure 10-3. Test Report in HTML or XML Format

13. Close the Execution window.
14. Select **Configure»Report Options**.
15. Select **ASCII Text File** in the Report Format ring control and set the **Result Filtering Expression** control back to `True` by selecting the **All Results** option.
16. Click **OK** to close the Report Options dialog box.

Using External Report Viewers

TestStand allows you to view the test report in external applications more suited for displaying and editing text, such as Microsoft Word or Microsoft Excel. TestStand refers to these external applications as external report viewers.

By default, the Windows operating systems can associate an application with a file extension. For example, by default, Microsoft associates the `.doc` file extension with the Microsoft WordPad application. If you install Microsoft Word on your system, the Word installer replaces the WordPad file type association with itself for the `.doc` file type.

1. Change the file extension of your report.
   a. Select **Configure»Report Options**.
   b. Select the **Report File Pathname** tab.
   c. Disable the **Use Standard Extension for Report Format** option.
   d. Type `doc` in the Extension string control. TestStand will now create test reports with a `.doc` file extension.
   e. Click **OK** to close the Report Options dialog box.
2. Configure TestStand to automatically launch the external viewer associated with the `.doc` file extension.
   a. Select **Configure»External Viewers**.
   b. Enable the **Automatically Launch Default External Viewers** option.
   c. Click **OK** to close the Configure External Viewers dialog box.
3. Execute the sequence by selecting **Execute»Test UUTs**.
4. Run through several iterations of the sequence.
5. Click **Stop** in the UUT Information dialog box to stop the execution.

   TestStand generates the text report and launches WordPad or Microsoft Word to display the test report.
6. Examine the test report and close the external report viewing application.
7. Change the report settings back, as follows:
   a. Select **Configure»Report Options**.
   b. On the **Contents** tab, select **HTML Document** or **XML Document** in the Report Format ring control.

c. Enable the **Use Standard Extension for Report Format** option on the **Report File Pathname** tab.

d. Click **OK** to close the Report Options dialog box.

8. Complete the following steps to change the external viewer settings back to their default settings:

   a. Select **Configure»External Viewers**.

   b. Disable the **Automatically Launch Default External Viewers** option.

   c. Click **OK** to close the Configure External Viewers dialog box.

If you want to define a file association independent of your operating system, use the Configure External Viewers dialog box. Refer to Chapter 6, *Database Logging and Report Generation*, in the *TestStand Reference Manual* for more information about external viewers.

Adding Custom Step Type Properties to a Report

Step types define a list of properties and behaviors for each step of that type. TestStand contains a set of predefined step types. If the functionality of these step types does not meet your needs, you can design new step types to suit your application. For more information about step types, refer to Chapter 11, *Type Concepts*, in the *TestStand Reference Manual*.

In this tutorial, you will create a new step type with a custom numeric *array property* and include this property in your test report.

Creating a Step Type

1. Open a new sequence by selecting **File»New Sequence File**.
2. Save the sequence as `CustomReport.seq` in the `<TestStand>\Tutorial` directory.
3. Choose **Sequence File Types** from the **View** ring control in the Sequence File window. The **Step Types** tab should be selected.
4. Right-click and select **Insert Step Type** from the context menu. Rename the step type `NumericArray`.
5. Insert a custom step property that is a numeric array. In the left tree view pane, expand the `NumericArray` step type and select the `Result` node.

   During execution, TestStand automatically collects the values of all properties within the Result container of a step. It stores these as elements of the `Locals.ResultList` array. You can include these values in your test report.

a. From the right list view pane, right-click the `Error` property and select **Insert Field»Array of»Number** from the context menu.

b. In the Array Bounds dialog box of the inserted property, type `49` in the **Upper Bound** field and click **OK**.

   When you write an array to this property, TestStand automatically resizes the array to the size of your one-dimensional array.

c. Rename the property `NumArray`.

d. Right-click the `NumArray` property and select **Properties** from the context menu.

e. Click **Advanced** in the NumArray Properties dialog box.

f. In the Edit Flags dialog box, enable `PropFlags_IncludeInReport` from the list of available flags. This flag instructs TestStand to add the value of the property to the report.

g. Click **OK** twice to close the Edit Flags and NumArray Properties dialog boxes.

6. Right-click `NumericArray` and choose **Properties** from the context menu.

   The NumericArray Properties dialog box allows you to configure the behavior of your step type including the default values of all step properties such as run options, looping options, expressions, and post actions. You can create substeps that call code modules before and after a user-defined code module. Substeps define standard actions that occur for every instance of a step type.

7. On the **General** tab, enter the following values into the corresponding control:

Default Step Name Expression	`"Numeric Array Step"`
Step Description Expression	`"My Description" + "%ModuleDescription"`

   Leave the other controls set to their default values.

8. Click **OK** to close the NumericArray Properties dialog box.

9. Save your sequence file by selecting **File»Save**.

   TestStand launches a dialog box to warn you that you are saving a sequence file with modified types. Click **OK to** close this dialog box.

You have just created a step type with a property that can receive a numeric array from a code module. By locating this property within the Result container of the step type and enabling its `PropFlags_IncludeInReport` flag, the numeric array for each instance of your step type is included in the report automatically.

TestStand features other step type capabilities beyond those that you used in this tutorial. Typical modifications of a step type might include one or more post substeps, or a status expression that evaluates the numeric array to determine whether the step passes or fails. The step type might also have one or more *Edit substep*s that allow you to configure the pass/fail criteria prior to executing the test sequence. In an instance of a step type a context menu item is created for each Edit substep. You can also create *code template*s to help users create code modules for use with the custom step types.

For more information about creating custom step types, refer to Chapter 13, *Creating Custom Step Types*, in the *TestStand Reference Manual*.

Creating an Instance of the Step Type

In this tutorial, you create an instance of your step type that calls a DLL code module, which returns a numeric array.

1. Open `<TestStand>\Tutorial\CustomReport.seq`.
2. Select `MainSequence` from the View ring control.
3. Select **C/C++ DLL** from the Adapter ring control.
4. In the Main step group of your `MainSequence`, right-click and select **Insert Step»NumericArray** from the context menu.

   Notice that the default step name and step description are the values you entered when you created the step type.
5. Right-click the `Numeric Array Step` and select **Specify Module** from the context menu.
6. Under the **Module** tab of the Edit C/C++ DLL Call dialog box, click the **File Browse** button and navigate to `<TestStand>\Tutorial\NumericArray.dll`. Click **OK**.
7. Select **GetNumericArray** from the Function Name ring control.
8. TestStand launches a dialog box to ask whether you want to use the parameter information stored in the DLL. Click **Yes**.

   TestStand launches another dialog box asking you to specify the number of elements for the array parameter of the GetNumericArray function.

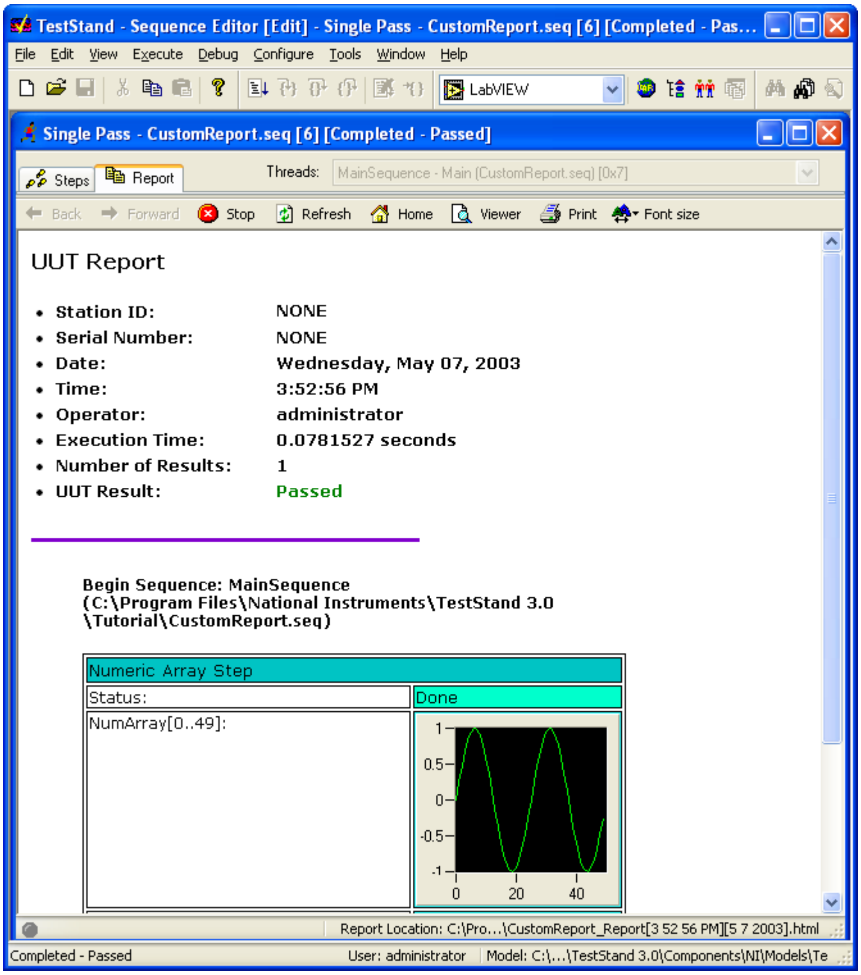
9. Type `50` in the Value column of the Dim 1 Size property and then click **OK** to close the dialog box.
10. In the Value Expression column of the measurements parameter, type the following expression:

    `Step.Result.NumArray`

    When TestStand returns from calling the GetNumericArray function, it will insert the value from the NumericArray output parameter into the `Result.NumArray` property of the step.
11. Click **OK to** save your settings and to return to the sequence editor.
12. Select **Configure»Report Options** and chance the **Include Array** option to **Insert Graph**. Change the **Report Format** option to **HTML** or **XML**.

    You are now ready to execute the sequence that calls the GetNumericArray DLL function.
13. Select **Execute»Single Pass** to execute the sequence.

    The array data will appear in the test report as illustrated in Figure 10-4.


TestStand - Sequence Editor [Edit] - Single Pass - CustomReport.seq [6] [Completed - Pas...
File Edit View Execute Debug Configure Tools Window Help
LabVIEW
Single Pass - CustomReport.seq [6] [Completed - Passed]
Steps
Report
Threads: MainSequence - Main (CustomReport.seq) [0x7]
Back Forward Stop Refresh Home Viewer Print Font size

UUT Report

- Station ID: NONE
- Serial Number: NONE
- Date: Wednesday, May 07, 2003
- Time: 3:52:56 PM
- Operator: administrator
- Execution Time: 0.0781527 seconds
- Number of Results: 1
- UUT Result: Passed

Begin Sequence: MainSequence
(C:\Program Files\National Instruments\TestStand 3.0
\Tutorial\CustomReport.seq)

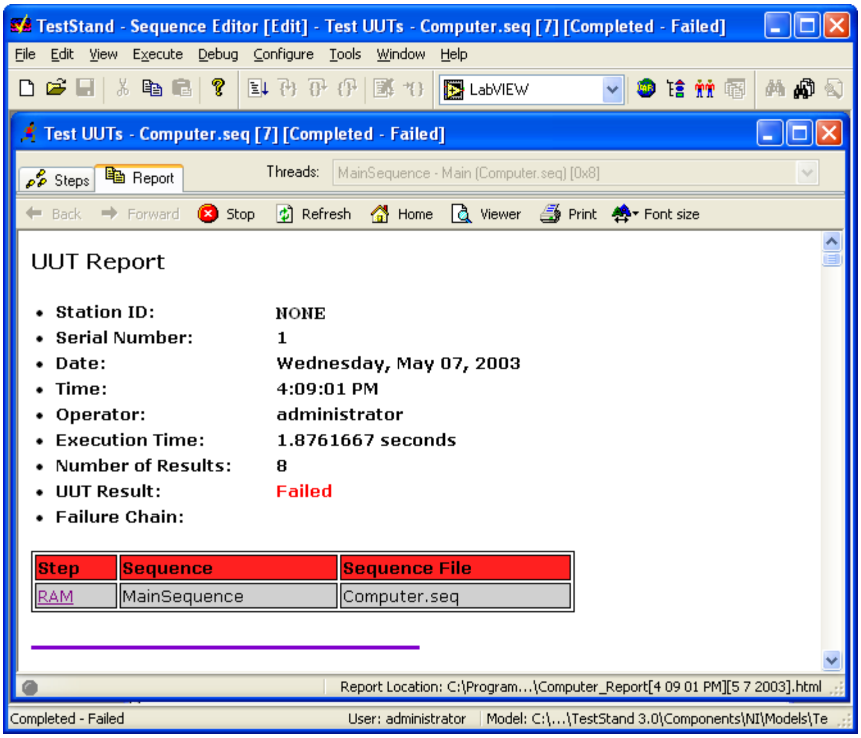
Numeric Array Step	
Status:	Done
NumArray[0..49]:	

1
0.5
0
-0.5
-1
0
20
40
Report Location: C:\Pro...\CustomReport_Report[3 52 56 PM][5 7 2003].html
Completed - Passed
User: administrator
Model: C:\...\TestStand 3.0\Components\NI\Models\Te

Figure 10-4. HTML or XML Report with Graph

Adding to a Report Using Callbacks

The default header for an HTML report appears as shown in Figure 10-5. In this tutorial, you add a logo to the header of the HTML report using a Report callback in the process model.


TestStand - Sequence Editor [Edit] - Test UUTs - Computer.seq [7] [Completed - Failed]
File Edit View Execute Debug Configure Tools Window Help
LabVIEW
Test UUTs - Computer.seq [7] [Completed - Failed]
Steps
Report
Threads: MainSequence - Main (Computer.seq) [0x8]
Back Forward Stop Refresh Home Viewer Print Font size

UUT Report

• Station ID:	NONE
• Serial Number:	1
• Date:	Wednesday, May 07, 2003
• Time:	4:09:01 PM
• Operator:	administrator
• Execution Time:	1.8761667 seconds
• Number of Results:	8
• UUT Result:	Failed
• Failure Chain:	

Step	Sequence	Sequence File
RAM	MainSequence	Computer.seq

Report Location: C:\Program...\Computer_Report[4 09 01 PM][5 7 2003].html

Completed - Failed User: administrator Model: C:\...\TestStand 3.0\Components\NI\Models\Te

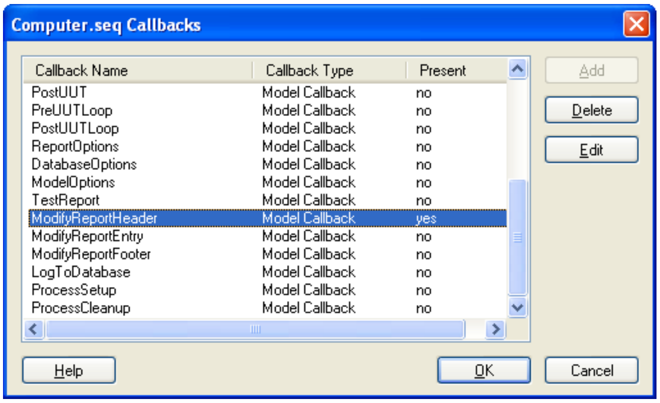
Figure 10-5. Default Header For An HTML Report

Note This tutorial assumes that the Report Format control in the Report Options dialog box is set to **HTML Document**.

1. Open `<TestStand>\Tutorial\Computer.seq`.
2. With the Sequence File window as the active window, select **Edit»Sequence File Callbacks to** open the Sequence File Callbacks dialog box.
3. Select the `ModifyReportHeader` callback.

4. Click **Add** to add the callback to the sequence file.

   The resulting dialog box is shown in Figure 10-6.



Callback Name	Callback Type	Present
PostUUT	Model Callback	no
PreUUTLoop	Model Callback	no
PostUUTLoop	Model Callback	no
ReportOptions	Model Callback	no
DatabaseOptions	Model Callback	no
ModelOptions	Model Callback	no
TestReport	Model Callback	no
ModifyReportHeader	Model Callback	yes
ModifyReportEntry	Model Callback	no
ModifyReportFooter	Model Callback	no
LogToDatabase	Model Callback	no
ProcessSetup	Model Callback	no
ProcessCleanup	Model Callback	no

Figure 10-6. Sequence File Callbacks Dialog Box

5. Click **Edit** to close the Sequence File Callbacks dialog box and edit the new `ModifyReportHeader` callback sequence. Notice that the ModifyReportHeader callback sequence is automatically opened in the Sequence File window.
6. Click the **Locals** tab and right-click in the right pane.
7. Select **Insert Local»String** from the context menu to insert a local string variable.
8. Rename the variable `AddToHeader`.
9. Double-click the `AddToHeader` variable.
10. Enter the following value in the Value field of the String Properties dialog box:

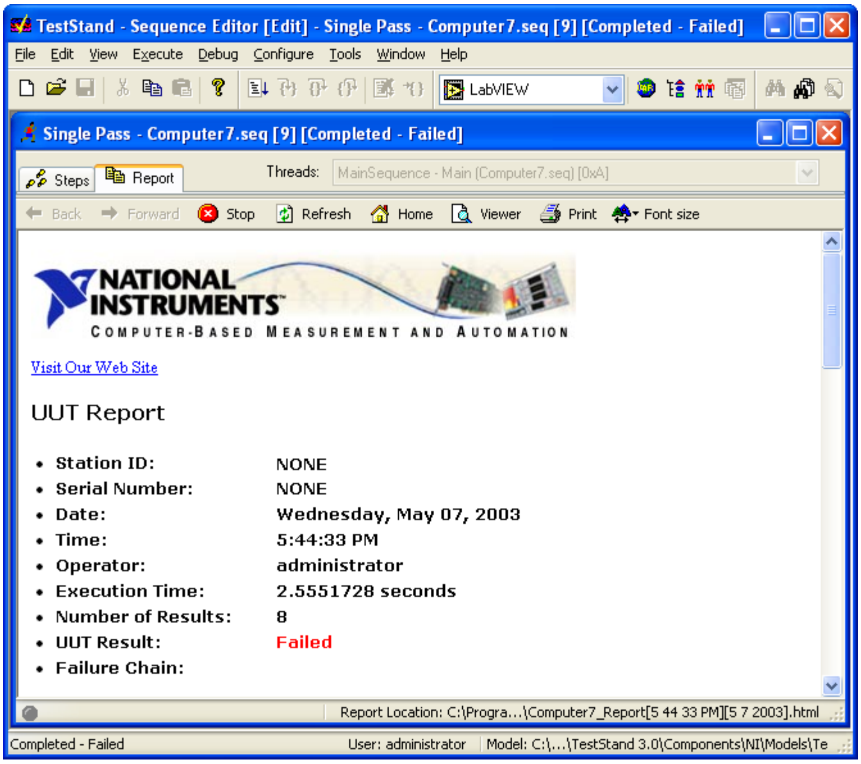
```
<IMG ALT='Logo Goes Here' SRC='Logo.jpg'><br></br>
<A HREF='http://www.ni.com'>Visit Our Web
  Site</A><br></br>
```

11. Click **OK** to close the String Properties dialog box.
12. Click the **Main** tab to display the Main step group, which is empty.
13. Right-click in the steps list and insert a Statement step by selecting **Insert Step»Statement** in the context menu.
14. Rename this step `Add Custom Logo`.
15. Right-click the `Add Custom Logo` step and select **Edit Expression** from the context menu.

16. Enter the following expression:

    `Parameters.ReportHeader = Locals.AddToHeader + Parameters.ReportHeader`

17. Click **OK** to close the Edit Statement Step dialog box.
18. Select **File»Save As** and save the sequence in the `<TestStand>\Tutorial` directory as `Computer7.seq`.
19. Select **Execute»Single Pass**.
20. Click **Done** in the Test Simulator dialog box.
21. After the execution completes, view the report and notice the new logo image at the top of the UUT report, as shown in Figure 10-7.


TestStand - Sequence Editor [Edit] - Single Pass - Computer7.seq [9] [Completed - Failed]
File Edit View Execute Debug Configure Tools Window Help
LabVIEW
Single Pass - Computer7.seq [9] [Completed - Failed]
Steps
Report
Threads:
MainSequence - Main (Computer7.seq) [0xA]
Back Forward Stop Refresh Home Viewer Print Font size
NATIONAL INSTRUMENTS
COMPUTER-BASED MEASUREMENT AND AUTOMATION
Visit Our Web Site

UUT Report

- Station ID: NONE
- Serial Number: NONE
- Date: Wednesday, May 07, 2003
- Time: 5:44:33 PM
- Operator: administrator
- Execution Time: 2.5551728 seconds
- Number of Results: 8
- UUT Result: Failed
- Failure Chain:

Report Location: C:\Progra...\Computer7_Report[5 44 33 PM][5 7 2003].html

Completed - Failed User: administrator Model: C:\...\TestStand 3.0\Components\NI\Models\Te

Figure 10-7. HTML Report with New Header

22. Close the Execution and Sequence File windows.

For more details about customizing reports, refer to Chapter 6, *Database Logging and Report Generation*, in the *TestStand Reference Manual*.

You have completed all of the tutorials in this manual. For more information about the TestStand concepts and components discussed in these tutorials, refer to the following TestStand documentation:

- *TestStand System and Architecture Overview Card*
- *TestStand Help*
- *TestStand Reference Manual*

For more information about using LabVIEW and the LabVIEW Adapter with TestStand, refer to *Using LabVIEW with TestStand*. For more information about using LabWindows/CVI and the LabWindows/CVI Adapter with TestStand, refer to *Using LabWindows/CVI with TestStand*.

A Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Online technical support resources include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit our extensive library of technical support resources available in English, Japanese, and Spanish at `ni.com/support`. These resources are available for most products at no cost to registered users and include software drivers and updates, a KnowledgeBase, product manuals, step-by-step troubleshooting wizards, conformity documentation, example code, tutorials and application notes, instrument drivers, discussion forums, a measurement glossary, and so on.
  - **Assisted Support Options**—Contact NI engineers and other measurement and automation professionals by visiting `ni.com/support`. Our online system helps you define your question and connects you to the experts by phone, discussion forum, or email.
- **Training**—Visit `ni.com/training` for self-paced tutorials, videos, and interactive CDs. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. to learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

A

abort	To stop an execution without running any of the Cleanup step groups in the sequences on the call stack. When you abort an execution, report generation does not occur.
active window	The window that user input affects at a given moment. The title of an active window is highlighted.
ActiveX server	Any executable code that makes itself available to other applications according to the ActiveX standard. ActiveX implies a client/server relationship in which the client requests objects from the server and asks the server to perform actions on the objects.
Adapter	*See* module adapter.
administrator	A user profile that usually contains all privileges for a test station.
Application Development Environment (ADE)	A programming environment such as LabVIEW, LabWindows/CVI, or Microsoft Visual C, in which you can create code modules and operator interfaces.
Application Programming Interface (API)	A set of classes, methods, and properties that you use to control a specific service, such as the TestStand Engine.
array property	A property that contains an array of single-valued properties of the same type.
ASCII	American Standard Code for Information Interchange.

B

breakpoint	An interruption in the execution of a program.
button	A dialog box item that, when selected, executes a command associated with the dialog box.

C

call stack	The chain of active sequences that are waiting for nested subsequences to complete.
callback	A sequence that is used to handle common tasks, such as serial number inquiry or report logging.
checkbox	A dialog box input that allows you to toggle between two possible options.
client sequence file	A sequence file that contains the Main sequence a process model invokes to test a UUT. Each client sequence file contains a sequence called `MainSequence`. The process model defines what is constant about your testing process, whereas the client sequence file defines the steps that are unique to the different types of tests you run.
code module	A program module, such as a Windows Dynamic Link Library (`.dll`) or LabVIEW VI (`.vi`), that contains one or more functions that perform a specific test or other action.
code template	A source file that contains skeleton code. The skeleton code serves as a starting point for the development of code modules for steps that use the step type.
Container property	A property that contains no values and typically contains multiple subproperties. Container properties are analogous to structures in C/C++ and to clusters in LabVIEW.
context menu	Menus accessed by clicking an object. Menu options pertain to that object specifically.
control	An input and output device for entering data that appears on a panel or window.
CPU	Central Processing Unit.

D

developer	A user profile that usually contains all privileges associated with operating, debugging, and developing sequences and sequence files, but cannot configure user privileges, report options, or database options.
dialog box	A prompt mechanism in which you enter additional information needed to complete a command.
DLL	Dynamic Link Library.

E

Edit substep	A substep that the engine calls when editing the step. You invoke the substep with the menu item that appears in the context menu above **Specify Module**. The Edit substep displays a dialog box in which the sequence developer edits the values of custom step properties. For example, the Edit Limits item appears in the context menu for Numeric Limit test steps, and the Edit Pass/Fail Source item appears in the context menu for Pass/Fail test steps.
engine	*See* test executive engine.
entry points	A sequence in the process model file that TestStand displays as a menu item, such as Test UUTs, Single Pass, and Report Options.
execution	An object that contains all the information TestStand needs to run a sequence, its steps, and any subsequences it calls. Typically, the TestStand Sequence Editor creates a new window for each execution.
Execution entry points	Sequences in a process model that run tests against a UUT. Execution entry points call the MainSequence callback in the client sequence file. The default process model contains two Execution entry points: Test UUTs and Single Pass. By default, Execution entry points appear in the Execute menu. Execution entry points appear in the menu only when the active window contains a sequence file that has a MainSequence callback.
execution pointer	A yellow pointer icon that shows the progress of execution by pointing to the currently executing step in the Steps tab.

Execution window	A window in the sequence editor that displays the steps an execution runs. When execution is suspended, the Execution window displays the next step to execute and provides single-stepping options. You can also view variables and properties in any active sequence context in the call stack.
expression	A formula that calculates a new value from the values of multiple variable or properties. In expressions, you can access all variables and properties in the sequence context that is active when TestStand evaluates the expression. The following is an example of an expression: `Locals.MidBandFrequency = (Step.HighFrequency + Step.LowFrequency) / 2`
external viewer	An application other than TestStand that you use to view test reports and sequence file documentation. For example, you can use Microsoft Internet Explorer and Notepad as external viewers for HTML and ASCII-text files.

G

global variable	TestStand defines two types of global variables: sequence file global variables and station global variables. Sequence file global variables are accessible by any sequence or step in the sequence file. Station global variables are accessible by any sequence file loaded on the station. The values of station global variables are persistent across different executions and even across different invocations of TestStand.

H

highlight	The way in which input focus appears on a TestStand screen; to move the input focus onto an item.

I

interactive mode	When you run steps by selecting one or more steps in a sequence and choosing the **Run Selected Steps** or **Loop Selected Steps** items in the context menu or menu bar. The selected steps in the sequence execute, regardless of any branching logic that the sequence contains. The selected steps run in the order in which they appear in the sequence.
internal viewer	The functionality in the TestStand Sequence Editor that you can use to view test reports. Access the internal viewer by selecting the Report tab in an Execution window.

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench. A program development application based on the G programming language and used commonly for test and measurement purposes.
local variables	Properties of a sequence that hold a value or additional subproperties. Only a step within the sequence can directly access the property value.

M

`MainSequence`	The sequence that initiates the tests on a UUT. The process model invokes the Main sequence as part of the overall testing process. The process model defines what is constant about your testing process, whereas the Main sequence defines the steps that are unique to the different types of tests you run.
menu bar	Horizontal bar that contains names of main menus.
method	Performs an operation or function on an object.
MFC	Microsoft Foundation Class Library.
modal dialog box	A dialog box that you must dismiss before you can operate other application windows.
Model callback	A mechanism which allows a sequence file to customize the default behavior of a sequence in the process model.
model sequence file	A special type of sequence file that contains process model sequences. The sequences within the sequence file direct the high-level sequence flow of an execution when testing a UUT.
module adapter	A component that the TestStand Engine uses to invoke code in another sequence or in a code module, such as LabVIEW. When invoking code in a code module, the adapter knows how to call it, and how to pass parameters to it.

N

nested	Called by another step or sequence. If a sequence calls a subsequence, the subsequence is nested in the invocation of the calling sequence.
normal execution	When you start an execution in the sequence editor by selecting the Run Sequence Name item or one of the Process Model entry points from the Execute menu.

O

object	A service that an ActiveX server makes available to clients.
operator	A user profile that usually contains all privileges associated with operating a test station, but cannot debug sequence executions, edit sequence files, or configure user privileges, station options, report options, and database options.
operator interface	A program that provides a graphical user interface (GUI) for executing sequences at a production station. Sometimes the sequence editor and operator interfaces are different aspects of the same program.

P

parent sequence	The calling sequence of a subsequence.
post actions	Actions that TestStand takes depending on the pass/fail status of the step or a custom condition the engine evaluates after executing a step. Post actions allow you to execute callbacks or jump to other steps after executing the step.
preconditions	A set of conditions for a step that must be `True` for TestStand to execute the step during the normal flow of execution in a sequence.
process model	A sequence file you designate that performs a standard series of operations before and after a test executive executes the sequence that performs the tests. Common operations include identifying the UUT, notifying the operator of pass/fail status, generating a test report, and logging results.
property	A container of information, which stores and maintains a setting or attribute of an object. A property can contain a single value, an array of values of the same type, or no value at all. A property can also contain any number of subproperties. Each property has a name.

R

RAM	Random-access memory.
ROM	Read-only memory.
root interactive execution	When you run selected steps from a Sequence File window in an independent execution. Root interactive executions do not invoke process models.
run-time error	An error condition that forces an execution to terminate. When the error occurs while running a sequence, TestStand jumps to the Cleanup step group, and the error propagates to any calling sequence up the call stack.

S

s	Seconds.
sequence	A series of steps that you specify for execution in a particular order. Whether and when a step is executed can depend on the results of previous steps.
sequence context	A TestStand object that contains references to all global variables and all local variables and step properties in active sequences. The contents of the sequence context changes depending on the currently executing sequence and step.
sequence editor	A program that provides a graphical user interface for creating, editing, and debugging sequences.
sequence file	A file that contains the definition of one or more sequences.
Sequence File window	A separate window within the sequence editor that a sequence file appears in.
sequence file global variables	Variables you can use to store data relevant to the entire sequence file. Each sequence and step in the sequence file can directly access these global variables.
station global variables	Variables that are persistent across different executions and even across different invocations of the sequence editor or operator interfaces. The TestStand Engine maintains the value of station global variables in a file on the run-time computer.

station model	A process model that you select to use for all sequence files for a station. The TestStand installation program establishes `SequentialModel.seq` as the default station model file. You can use the Station Options dialog box to select a different station model.
step	Any action, such calling a code module to perform a specific test, that you can include within a sequence of other actions.
step group	A set of steps in a sequence. A sequence contains the following groups of steps: Setup, Main, and Cleanup. When TestStand executes a sequence, the steps in the Setup step group execute first, the steps in the Main step group execute next, and the steps in the Cleanup step group last.
step property	A property of a step.
step result	A container property that contains a copy of the subproperties from the Result property of a step and additional execution information such as the name of the step and its position in the sequence. TestStand automatically creates a step result as each step executes and places the step result into a result list which TestStand uses to generate its reports.
step status	A string value that indicates the status of a step in an execution. Every step in TestStand has a Result.Status property. Although TestStand imposes no restrictions on the values to which the step or its code module can set the status property, TestStand and the built-in step types use and recognize a predefined set of values.
step type	A component that defines a set of custom step properties and standard behavior for each step of that type. All steps of the same type have the same properties, but the values of the properties can differ. Step types define their standard behaviors using substeps.
subsequence	A sequence that another sequence calls. You specify a subsequence call as a step in the calling sequence.
substep	Actions that a step type performs for a step besides calling the code module. You define a substep by selecting an adapter and specifying a module call. TestStand defines three different types of substeps: Edit substep, Pre-Step substep, and Post-Step substep.

T

technician	A user profile that usually contains all privileges associated with operating, and debugging sequences and sequences files, but cannot edit sequence files or configure user privileges, station options, report options, or database options.
template	*See* code template.
terminate	To stop an execution by halting the normal execution flow, and running all the Cleanup step groups in the sequences on the call stack.
test executive engine	A module or set of modules that provide an API for creating, editing, executing, and debugging sequences. A sequence editor or operator interface uses the services of a test executive engine.

U

Unit Under Test (UUT)	The device or component that you are testing.
User Manager	The component of the TestStand Engine that maintains a list of users, their user names and passwords, and their privileges. You can access the user manager from the User Manager window in the sequence editor.

V

variable	A property that you can freely create in certain contexts. You can have variables that are global to a sequence file or local to a particular sequence. You can also have station global variables.
VI	Virtual instrument.

W

window	A working area that supports specific tasks related to developing and executing programs.

Index