TestStand™

Using LabVIEW™ with TestStand

NATIONAL INSTRUMENTS™

July 2003 Edition
Part Number 323200A-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.
♦	The ♦ symbol indicates that the following text applies only to a specific product, a specific operating system, or a specific software version.
💡	This icon denotes a tip, which alerts you to advisory information.
📝	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click 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

This chapter discusses how TestStand and LabVIEW work together in a test system.

The Role of LabVIEW in a TestStand-Based System

TestStand is a test management environment that you use to organize and execute code modules written in a variety of languages and application development environments (ADEs), including LabVIEW. It handles core test management functionality such as the definition and execution of the overall testing process, user management, report generation, database logging, and more. TestStand can work in a variety of different testing scenarios and environments because it allows extensive customization of components like the process model, step types, and operator interfaces. You can use LabVIEW to accomplish much of this customization.

You typically use LabVIEW to customize your test system in the following ways:

- Create code modules—such as tests and actions—that TestStand can call using the LabVIEW Adapter
- Create custom operator interfaces for your test system
- Create custom step types

Code Modules

TestStand can call LabVIEW VIs with a variety of connector pane configurations. It can also call VIs that reside on either the same machine as TestStand or on other network computers, including computers running the LabVIEW Real-Time (RT) Module.

TestStand can pass data to the VIs it calls and store the data that the VI returns. Additionally, VIs that TestStand calls can access the complete TestStand application programming interface (API) for advanced applications.

Operator Interfaces

You can use LabVIEW to build custom user interfaces for your test systems. Typically, these custom user interfaces are operator interfaces designed for use in production test systems. The full power of the LabVIEW development environment allows you to customize these interfaces to meet your exact requirements. Refer to Chapter 9, *Creating Custom Operator Interfaces*, in the *TestStand Reference Manual*, for more information about creating custom operator interfaces.

Custom Step Types

You can use LabVIEW to create VIs that you call from custom step types. These VIs can implement editing dialog boxes and other features of custom step types. Refer to Chapter 13, *Creating Custom Step Types*, in the *TestStand Reference Manual* for more information about custom step types.

Version Compatibility

TestStand 3.0 is compatible with LabVIEW 6.1 and later. Because of the increased support for TestStand integration provided by LabVIEW 7.0, not all of the TestStand 3.0 LabVIEW Adapter functionality is available to LabVIEW 6.1 users.

Table 1-1 itemizes these differences in functionality.

Table 1-1. Version Compatibility

TestStand Functionality	**LabVIEW 6.1**	**LabVIEW 7.0 or later**
Call VIs with arbitrary connector panes	No	Yes
Call VIs on remote computers	No	Yes
Run VIs in the LabVIEW Run-Time Engine	No	Yes
Create User Interfaces using the TestStand UI Controls	No	Yes. Note that this functionality requires the LabVIEW Full or Professional Development System.
Call TestStand VIs from versions of TestStand prior to 3.0 and LabVIEW Test Executive VIs	Yes	Yes

Table 1-1. Version Compatibility (Continued)

TestStand Functionality	LabVIEW 6.1	LabVIEW 7.0 or later
Open VIs for editing from TestStand	Yes	Yes
Create VIs from TestStand	Yes	Yes
Debug VIs (step in/step out) from TestStand	Yes	Yes
Run VIs using the LabVIEW development system	Yes	Yes
Run VIs using a LabVIEW executable	Yes	Yes
Create User Interfaces using the TestStand API	Yes	Yes
Program with the TestStand API	Yes	Yes

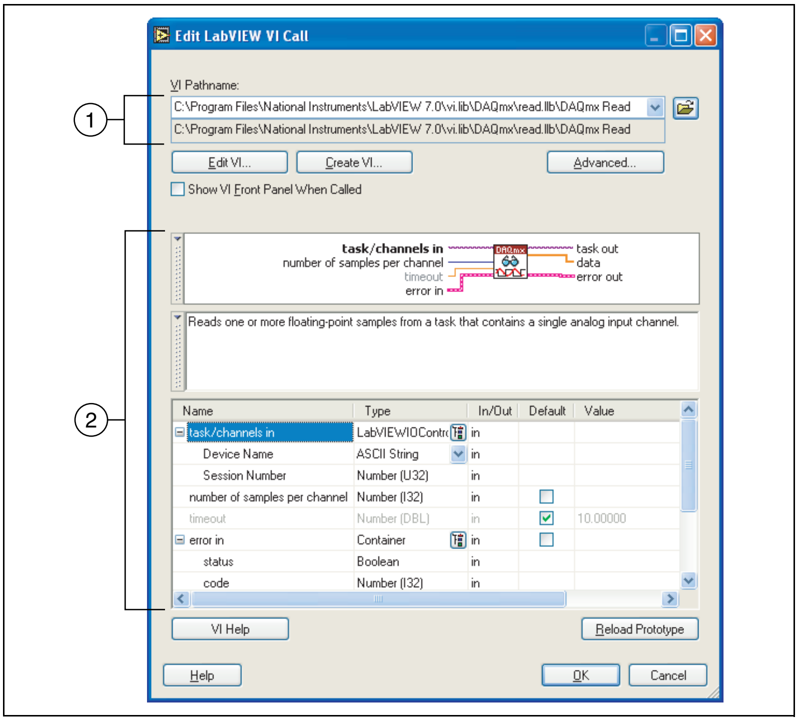
2 Calling LabVIEW VIs from TestStand

This chapter discusses how to call LabVIEW VIs from TestStand using the LabVIEW Adapter.

Introduction to the Edit LabVIEW VI Call Dialog Box

Configure calls to LabVIEW VIs using the Edit LabVIEW VI Call dialog box, which is shown in Figure 2-1. To launch this dialog box, select **Specify Module** from the context menu of any step that uses the LabVIEW Adapter.

The Edit LabVIEW VI Call dialog box is divided into two main sections, which are illustrated in Figure 2-1.



C:\Program Files\National Instruments\LabVIEW 7.0\vi.lib\DAQmx\read.llb\DAQmx Read
C:\Program Files\National Instruments\LabVIEW 7.0\vi.lib\DAQmx\read.llb\DAQmx Read

☐ Show VI Front Panel When Called

Reads one or more floating-point samples from a task that contains a single analog input channel.

Name	Type	In/Out	Default	Value
⊟ task/channels in	LabVIEWIOContr	in		
Device Name	ASCII String	in		
Session Number	Number (U32)	in		
number of samples per channel	Number (I32)	in	☐	
timeout	Number (DBL)	in	✓	10.00000
⊟ error in	Container	in	☐	
status	Boolean	in		
code	Number (I32)	in		

1 Path and Execution Section	2 Description and Connector Pane Section

Figure 2-1. Edit LabVIEW VI Call Dialog Box

Path and Execution Section

The Path and Execution section of the Edit LabVIEW VI Call dialog box contains the pathname to the VI that the adapter will execute, as well as buttons for editing the VI in LabVIEW, creating a new VI based on the code template for the current step type, and opening the Advanced Settings dialog box. You can also use this section to specify whether LabVIEW displays the front panel of the VI when it is called by TestStand.

Description and Connector Pane Section

The Description and Connector Pane section of the Edit LabVIEW VI Call dialog box contains specific information about the VI to call, including complete information about the controls and indicators that are wired to the connector pane of the VI.

As shown in Figure 2-1, the Description and Connector Pane section is divided into the following three areas: the VI Context Help Picture, the VI Description, and the VI Parameter Table.

- **VI Context Help Picture**—Displays the context help picture of the VI as shown in the LabVIEW Context Help window. When you click on any label or connector inside the VI icon, the parameter control will automatically select that parameter. The connector pane also blinks to let you know which parameter is selected.
- **VI Description**—Displays the description of the VI from the Documentation page of the LabVIEW VI Properties dialog box. If there is no description, this control is hidden.
- **VI Parameter Table**—Contains information about each control or indicator wired to the connector pane of the VI. These are the *parameters* of the VI.

  The VI Parameter Table, in turn, contains the following information for each parameter:

  – **Name**—Caption text for the control or indicator. If there is no caption, this field contains the label text.

  – **Type**—LabVIEW data type for the control or indicator. Refer to Chapter 4, *Using LabVIEW Data Types with TestStand*, for more information about how LabVIEW data types map to TestStand data types.

  – **In/Out**—Specifies whether the parameter is an input (control) or an output (indicator).

  – **Default**—Specifies whether TestStand uses the default value of the control for any parameters. If the terminal on the VI is marked Required, this option is not available.

  – **Value**—A TestStand expression. For input parameters, TestStand passes the result of this expression to the VI, unless the checkbox in the Default column is enabled. For output parameters, TestStand stores the data that the VI outputs in the location specified by this parameter.

- **VI Help**—Displays the help associated with the VI.
- **Reload Prototype**—Allows you to refresh the parameter information for the VI.

Note Use the **Help** button at the bottom of the dialog box to access the *TestStand Help*, which provides additional information about the Edit LabVIEW VI Call dialog box.

Creating and Configuring a New Step Using the LabVIEW Adapter

In this tutorial, you will learn how to insert a new step that uses the LabVIEW Adapter and then configure that step to call a test VI.

1. Launch the TestStand Sequence Editor.
2. Select **LabVIEW Adapter** from the Adapter ring control.
3. Right-click inside the new Sequence File window and insert a new Pass/Fail step.
4. Rename the new step `LabVIEW Pass/Fail Test`.
5. Right-click the `LabVIEW Pass/Fail Test` step and select **Specify Module** from the context menu to launch the Edit LabVIEW VI Call dialog box.

♦ Proceed to Step 10 if you are using LabVIEW 6.1.

6. Click the **File Browse** button and select the following file: `<TestStand>\Tutorial\LabVIEW Pass-Fail Test.vi`.

   After you select the file, TestStand launches the File Not Found dialog box to indicate that the file does not reside in any of the TestStand search directories.
7. Select **Use a relative path for the file you selected** and click **OK**.

   TestStand now reads the description and connector pane information from the VI and updates the Description and Connector Pane section of the Edit LabVIEW VI Call dialog box.

   You can now configure the data to pass to and from the VI.
8. In the Value column of the VI Parameter Table, type `Step.Result.PassFail` for the `PASS/FAIL Flag` output parameter. Then, type `Step.Result.ReportText` in the Value column for the `Report Text` output parameter.

When TestStand calls the VI, it will place the value that the VI outputs in the **PASS/FAIL Flag** and **Report Text** indicators into the `Result.PassFail` and `Result.ReportText` properties of the step.

9. For the **error out** output parameter, notice that TestStand automatically fills in the Value column with the `Step.Result.Error` property.

Note By default, if a VI has the standard LabVIEW **error out** cluster as an output parameter, TestStand automatically passes its value into the `Step.Result.Error` property for the step. You can also update the value manually.

♦ Steps 10 and 11 are for LabVIEW 6.1 users only. Proceed to Step 12 if you are using LabVIEW 7.0 or later.

10. Click the **File Browse** button and select the following file: `<TestStand>\Tutorial\LabVIEW Pass-Fail Test(Legacy).vi`.

    After you select the file, TestStand launches the File Not Found dialog box to indicate that the file does not reside in any of the TestStand search directories.

11. Select **Use a relative path for the file you selected** and click **OK**.

    TestStand now reads the description and connector pane information from the VI and updates the Description and Connector Pane section of the Edit LabVIEW VI Call dialog box.

    Notice that TestStand automatically fills in the Value column for the **Test Data** and **error out** output parameters to return the data to the correct subproperties of the Result property for the step.

12. Click **OK** to exit the Edit LabVIEW VI Call dialog box and save the settings.

13. Select **File»Save As** and save the sequence file as `<TestStand>\Tutorial\Call LabVIEW VI.seq`.

14. Select **Execute»Single Pass** to run the sequence file using the Single Pass Execution entry point.

    When the execution is complete, the resulting report shows that the step passed. The VI always returns `True` as its Pass/Fail output parameter.

You have completed this tutorial. In the next chapter, you will learn how to create, edit, and debug VIs from TestStand.

3

Creating, Editing, and Debugging LabVIEW VIs from TestStand

This chapter discusses how to use the LabVIEW Adapter to create new VIs that you can call from TestStand, as well as how to edit and debug existing VIs.

All of the tutorials in this chapter require that you have the LabVIEW development system and TestStand installed on the same computer. You must also have TestStand configured to run VIs using the LabVIEW development system. Refer to Chapter 5, *Configuring the LabVIEW Adapter*, for more information about selecting the LabVIEW server that TestStand uses to run VIs.

Creating a New VI from TestStand

In this tutorial, you will learn how to create a new VI from TestStand.

1. Launch the TestStand Sequence Editor.
2. Open the following sequence file, which you created in the *Creating and Configuring a New Step Using the LabVIEW Adapter* section of Chapter 2, *Calling LabVIEW VIs from TestStand*, of this manual: `<TestStand>\Tutorial\Call LabVIEW VI.seq`.
3. Select **LabVIEW Adapter** from the Adapter ring control.
4. Insert a Numeric Limit Test step after the `LabVIEW Pass/Fail Test` step, and rename it `LabVIEW Numeric Limit Test`.
5. Right-click the new step and select **Specify Module** from the context menu.
6. Click **Create VI** to create a new VI.

♦ Step 7 is for LabVIEW 6.1 users only. Proceed to Step 8 if you are using LabVIEW 7.0 or later.

7. In the Optional Parameters dialog box, click **OK** to create the VI with no input parameters.
8. In the File dialog box that launches, browse to the `<TestStand>\Tutorial` subdirectory and type `LabVIEW Numeric Limit Test.vi` in the **File Name** control.
9. Click **OK** in the File dialog box.

   TestStand creates a new VI based on the available code templates for the TestStand Numeric Limit Test and then opens that VI in LabVIEW.
10. Open the block diagram for the new VI.
11. Complete this step according to the version of LabVIEW you are using:
    - If you are using LabVIEW 7.0 or later, right-click the **Numeric** indicator terminal and select **Create»Constant**. In the new constant, type the number `5.23`.
    - If you are using LabVIEW 6.1, right-click the **Numeric Measurement** input to the `TestStand - Create Test Data Cluster.vi` and select **Create»Constant**. In the new constant, type the number `5.23`.
12. Save and close the VI.

    In the Edit LabVIEW VI Call dialog box, notice that TestStand has automatically filled in the output parameters for the VI based on the information stored in the code template for the Numeric Limit Test step type.
13. Click **OK** to close the Edit LabVIEW VI Call dialog box and save the settings.
14. Select **File»Save As** and save the sequence file as `<TestStand>\Tutorial\Call LabVIEW VI 2.seq`.
15. Select **Execute»Single Pass** to run the sequence file using the Single Pass Execution entry point.

    When the execution is complete, the resulting report shows that the step passed with a numeric measurement of 5.23.

Note For more information about step types and code templates, refer to Chapter 13, *Creating Custom Step Types*, in the *TestStand Reference Manual*. For more information about creating VIs from TestStand, refer to Chapter 5, *Configuring the LabVIEW Adapter*, of this manual.

Editing an Existing VI from TestStand

In this tutorial, you will learn how to edit an existing VI from TestStand.

1. Launch the TestStand Sequence Editor.
2. Open the following sequence file: `<TestStand>\Tutorial\Call LabVIEW VI 2.seq`.
3. Right-click the `LabVIEW Pass/Fail Test` step and select **Edit Code**.

   LabVIEW becomes the active application in which `LabVIEW Pass-Fail Test.vi` is open and in an editable state.
4. Open the block diagram for the VI.
5. Change the Boolean constant to `False`.
6. Save and close the VI.
7. Select **Execute»Single Pass** to run the sequence file using the Single Pass Execution entry point.

   When the execution is complete, the resulting report shows that the step failed. The VI now returns `False` in the **Pass/Fail** indicator.

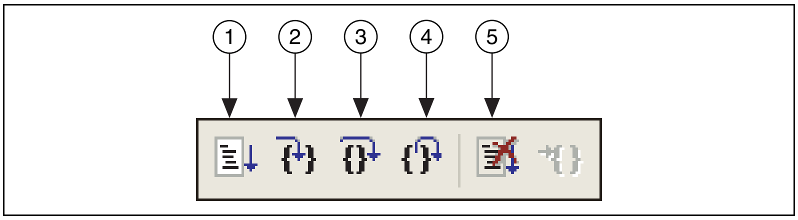
Debugging a VI in TestStand

In this tutorial, you will learn how to debug a VI that you call from TestStand using the LabVIEW Adapter.

1. Launch the TestStand Sequence Editor.
2. Open the following sequence file: `<TestStand>\Tutorial\Call LabVIEW VI.seq`.
3. Place a breakpoint at the `LabVIEW Pass/Fail Test` step by clicking to the left of the step.

   The **Stop** icon becomes visible to the left of the step when the breakpoint is set.
4. Select **Execute»Run MainSequence** to start an execution of `MainSequence`.

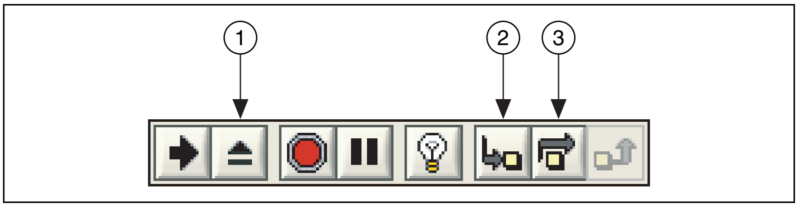
   The execution starts and then pauses at the breakpoint.
5. When the execution pauses, click **Step Into** on the Sequence Editor toolbar, which is shown in Figure 3-1.



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

Figure 3-1. Sequence Editor Toolbar

LabVIEW becomes the active application, in which the LabVIEW Pass-Fail Test VI is open and in a suspended state. The toolbar of the suspended VI is shown in Figure 3-2.



1	Return to Caller	2	Step Into	3	Step Over

Figure 3-2. Toolbar of the Suspended VI in LabVIEW

6. Open the block diagram of the VI.
7. Click **Step Into** or **Step Over** to begin single-stepping through the VI.
8. When you have finished single-stepping through the VI, click **Return to Caller** to return to TestStand. The execution then pauses at the next step in the sequence.
9. Select **Debug»Resume** in TestStand to complete the execution.

Note You can run the VI multiple times when the execution is in a suspended state. However, only the results from the last run will be passed back to TestStand when you finish debugging.

You have completed this tutorial. In the next chapter, you will learn how TestStand passes different types of data to and from LabVIEW VIs.

4 Using LabVIEW Data Types with TestStand

This chapter describes how TestStand converts LabVIEW data to and from its own data types.

Data Type Conversion

TestStand provides four basic built-in data types: number, string, Boolean, and object reference. TestStand also provides several standard named data types including Path, Error, LabVIEWAnalogWaveform, and others. You can create container data types that hold any number of other data types. TestStand containers are analogous to LabVIEW clusters.

LabVIEW has a greater variety of built-in data types than TestStand. For this reason, TestStand converts LabVIEW data types in certain ways when calling VIs. Table 4-1 describes how TestStand handles the various LabVIEW data types.

Table 4-1. TestStand Equivalents for LabVIEW Data Types

LabVIEW Data Type	TestStand Data Type
Real number (U8, U16, U32, I8, I16, I32, SGL, DBL, or EXT)	Number TestStand does not support extended-precision (EXT) floating point numbers and will convert any EXT numbers from LabVIEW into double-precision (DBL) numbers.
Complex number (CSG, CDB, or CXT)	Number TestStand maps each part of the complex number to separate TestStand Number properties. Refer to the information above about how TestStand converts EXT numbers.

Table 4-1. TestStand Equivalents for LabVIEW Data Types (Continued)

LabVIEW Data Type	TestStand Data Type
Enum (U32, U16, or U8)	Number For input parameters, the Value column of the Edit LabVIEW VI Call dialog box displays a ring containing the items in the LabVIEW Enum.
String	String Refer to the *Calling VIs with String Parameters* section for more information about the String data type.
Path	Path or String
ActiveX Control, Automation Refnum, or .NET Refnum	Object reference You cannot pass references to .NET objects that you create outside of LabVIEW, such as with the TestStand .NET Adapter, to LabVIEW VIs. You can store references to .NET objects that you create in LabVIEW within TestStand properties and then pass them to other LabVIEW VIs, provided that the objects are *marshallable by ref*.
Waveform	LabVIEWAnalogWaveform
Digital Waveform	LabVIEWDigitalWaveform
Digital Data	LabVIEWDigitalData
Picture	String You must select **Binary String** in the Edit LabVIEW VI Call dialog box. Refer to the *Calling VIs with String Parameters* section for more information about the String data type.
Refnum (File I/O, VI, Menu, Queue, TCP connection, and so on)	Number References to internal LabVIEW objects cannot be used inside TestStand or in other types of code modules. You can only store references to LabVIEW objects in TestStand properties and then pass them to other VIs.

Table 4-1. TestStand Equivalents for LabVIEW Data Types (Continued)

LabVIEW Data Type	TestStand Data Type
Timestamp	String
Error I/O	Error If a VI contains the standard **error out** cluster as an output parameter, TestStand automatically detects it and maps the output to Step.Result.Error.
Array of *x*	Array of TestStand (*x*)
Variant	Anything
Cluster	Container Refer to the *Calling VIs with Cluster Parameters* section for more information about the Container data type in TestStand.
I/O Data Types (DAQmx Task Name, DAQmx Channel Name, VISA Resource Name, IVI Logical Name, FieldPoint IO Point, or Motion Resource)	LabVIEWIOReference
IMAQ Session	Number
Other I/O data types (DAQmx Physical Channel Name, Terminal Name, Analog Trigger Source, Scale Name, Device Name, or Switch Name)	String

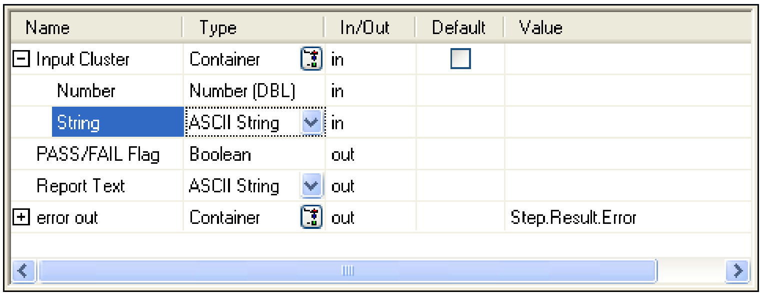
Note References to external objects, such as ActiveX objects or VISA sessions, can be used between different types of code modules.

Calling VIs with String Parameters

When you configure calls to VIs that have strings as parameters, you can specify whether TestStand *escapes* the string data when reading it from the VI or *unescapes* the string data when passing it to the VI. This option is necessary because, although LabVIEW strings can contain binary data including NULL characters, TestStand strings cannot contain NULL characters.

Use the ring control in the Type column of the VI Parameter Table for String parameters, shown in Figure 4-1, to select ASCII String or Binary String.

Note The default value of the ring control in the Type column for string parameters is ASCII String. TestStand does not modify the values of ASCII strings that it passes to or from VIs.



Name	Type	In/Out	Default	Value
⊟ Input Cluster	Container	in	☐	
Number	Number (DBL)	in		
String	ASCII String	in		
PASS/FAIL Flag	Boolean	out		
Report Text	ASCII String	out		
⊞ error out	Container	out		Step.Result.Error

Figure 4-1. VI Parameter Table for String Parameters

To store a LabVIEW string that contains binary data in a TestStand property, select **Binary String** in the Type column for the String parameter of the VI. TestStand then escapes the string before storing it, substituting hexadecimal codes for the unprintable characters in the string, such as NULL.

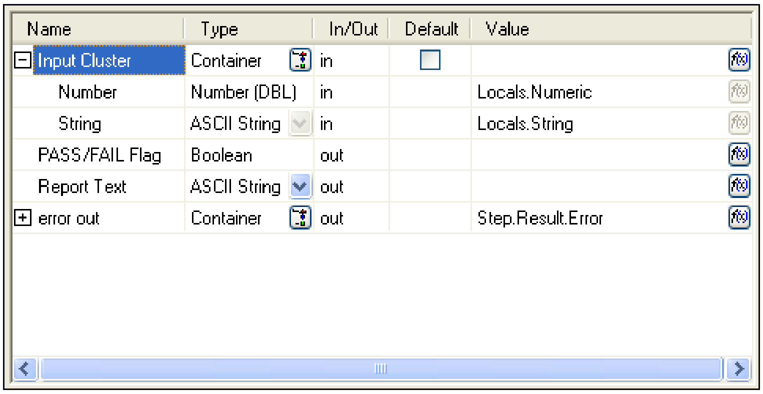
To pass a string that has been escaped to a LabVIEW VI, select **Binary String** in the Type column for the String parameter of the VI. TestStand unescapes the string before passing it to the VI, substituting the correct character values for the hexadecimal values in the escaped string.

Calling VIs with Cluster Parameters

When you configure calls to VIs that use clusters as parameters, you have two options for how to map those clusters in TestStand. You can specify that each cluster element maps to a different TestStand expression, or you can specify that a TestStand data type maps to the entire LabVIEW cluster. TestStand can also help you create a new custom data type that matches a LabVIEW cluster.

Specifying Each Cluster Element Individually

To configure each cluster element individually, specify a different TestStand expression for each element of the cluster. For example, Figure 4-2 illustrates a VI Parameter Table in which the data source for each element of the **Input Cluster** cluster is a different local variable.



Name	Type	In/Out	Default	Value
⊟ Input Cluster	Container	in	☐	
Number	Number (DBL)	in		Locals.Numeric
String	ASCII String	in		Locals.String
PASS/FAIL Flag	Boolean	out		
Report Text	ASCII String	out		
⊞ error out	Container	out		Step.Result.Error

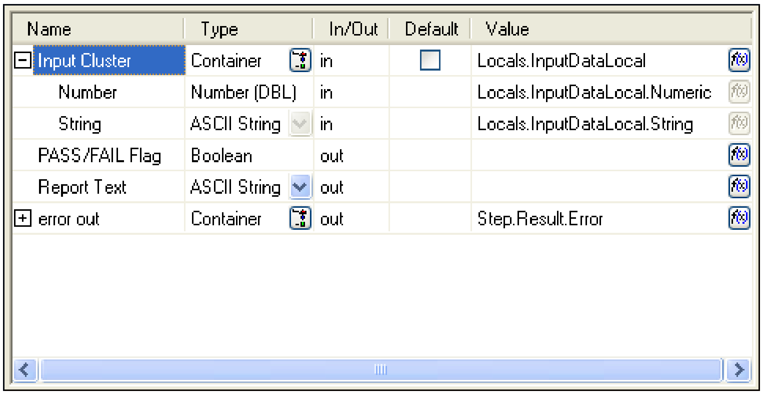
Figure 4-2. Input Cluster Cluster Data Sources

Passing Existing TestStand Container Variables to LabVIEW

Instead of passing each cluster element individually, as described in the previous section, you can also create a TestStand custom data type that matches your LabVIEW cluster.

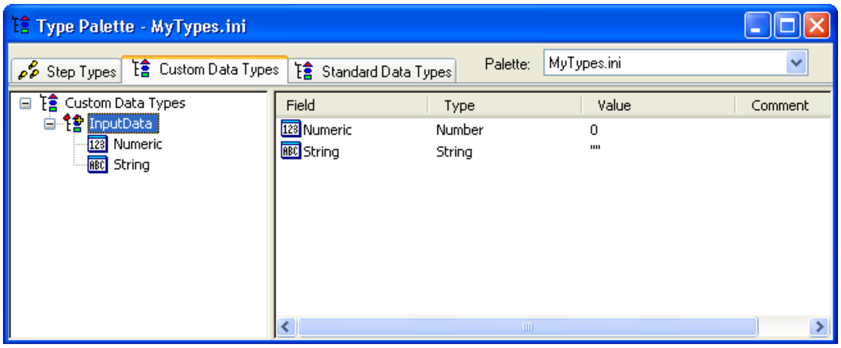
Use the **Cluster Passing** tab of the Type Properties dialog box for the new data type to specify how TestStand maps subproperties to elements in a LabVIEW cluster. Then, when you specify the data to pass for a cluster parameter, you only need to specify an expression that evaluates to data with the new data type. Refer to the *TestStand Help* for more information about the Type Properties dialog box.

Figure 4-3 illustrates how the data being passed to Input Cluster is a local variable called InputDataLocal, of the type InputData, which is shown in Figure 4-4.



Name	Type	In/Out	Default	Value
⊟ Input Cluster	Container	in	☐	Locals.InputDataLocal
Number	Number (DBL)	in		Locals.InputDataLocal.Numeric
String	ASCII String	in		Locals.InputDataLocal.String
PASS/FAIL Flag	Boolean	out		
Report Text	ASCII String	out		
⊞ error out	Container	out		Step.Result.Error

Figure 4-3. InputDataLocal Local Variable

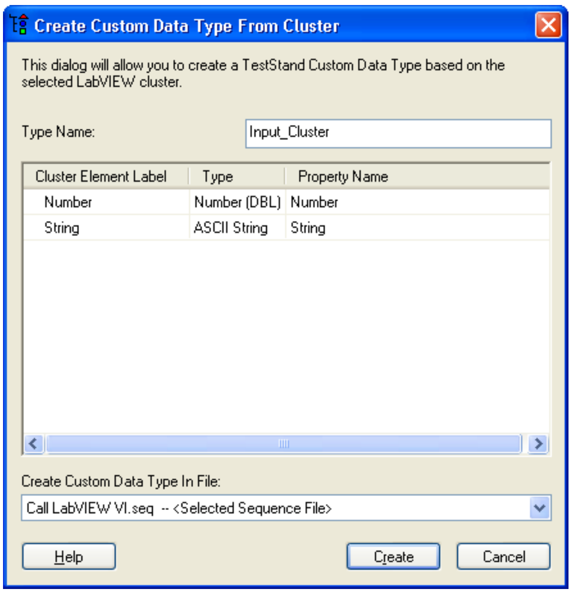


Field	Type	Value	Comment
Numeric	Number	0	
String	String	""	

Figure 4-4. TestStand Custom InputData Data Type

Creating a New Custom Data Type

If you have an existing LabVIEW cluster and want to create a TestStand custom data type that matches that cluster, TestStand automates this process with the Create Custom Data Type From Cluster dialog box, which is shown in Figure 4-5. This dialog box allows you to create a TestStand custom data type, such as a container, that is analogous to a LabVIEW cluster. Launch this dialog box by clicking the **Type Palette** button, which is located in the Type column of the VI Parameter Table in the Edit LabVIEW VI Call dialog box for those parameters that are clusters.


Create Custom Data Type From Cluster

This dialog will allow you to create a TestStand Custom Data Type based on the selected LabVIEW cluster.

Type Name: Input_Cluster

Cluster Element Label	Type	Property Name
Number	Number (DBL)	Number
String	ASCII String	String

Create Custom Data Type In File: Call LabVIEW VI.seq -- <Selected Sequence File>

Figure 4-5. Create Custom Data Type From Cluster Dialog Box

Specify the name of the TestStand custom data type you want to create in the Type column. Then, in the table's Property Name column, specify the names of the subproperties that map to the elements of the cluster. Finally, specify where TestStand should create the type in the Create Custom Data Type In File ring control.

Refer to Chapter 11, *Type Concepts*, in the *TestStand Reference Manual* for more information about where TestStand stores custom data types.

Creating TestStand Data Types from LabVIEW Clusters

In this tutorial, you will learn how to create a TestStand data type that matches a LabVIEW cluster.

Note You must have LabVIEW 7.0 or later installed in order to complete this tutorial.

1. Launch the TestStand Sequence Editor.
2. Open the following sequence file:
   `<TestStand>\Tutorial\Call LabVIEW VI 2.seq`.
3. Select **LabVIEW Adapter** from the Adapter ring control.

4. Insert a new Pass/Fail Test step into the Main step group of `MainSequence` after the `LabVIEW Numeric Limit Test` step.
5. Rename the step `Pass Container to VI`.
6. Right-click the new step and select **Specify Module** from the context menu.
7. Click the **File Browse** button and select the following file: `<TestStand>\Tutorial\VI with Cluster Input.vi`.

   Notice that the VI has a control labeled **Input Cluster**. This VI outputs a string containing the string element of the `Input Cluster` parameter in its **Report Text** indicator.
8. Click the **Type Palette** button in the Type column of the `Input Cluster` parameter to launch the Create Custom Data Type From Cluster dialog box, which you use to create a TestStand custom data type that matches the LabVIEW cluster.

   TestStand automatically maps fields in this cluster to subproperties in a container, Input_Cluster, that is a new TestStand data type. You can rename the data type and subproperties as necessary and specify where TestStand stores the new data type.

   Refer to the *Creating a New Custom Data Type* section for more information about the Create Custom Data Type From Cluster dialog box. For more information about custom data types, refer to the *TestStand Help* and to Chapter 12, *Standard and Custom Data Types*, in the *TestStand Reference Manual*.
9. In the Create Custom Data Type From Cluster dialog box, click **Create** to accept the automatically assigned values and to create the data type in the current sequence file.
10. In the Value column for the `Input Cluster` input parameter, click the **Expression Browse** button to open the Expression Browser dialog box.
11. Right-click **Locals** and select **Insert Types»Input_Cluster** to create a local variable of type `Input_Cluster`. Rename the local variable `ContainerData`.
12. Right-click each subproperty and select **Properties** to launch the Properties dialog box for the selected property. Initialize the values of the subproperties as follows:
    a. In the **Value** field for the `Number` subproperty, type `23`.
    b. In the **Value** field for the `String` subproperty, type `My String Data`.

13. In the Expression Browser dialog box, enter `Locals.ContainerData` in the Expression field and then click **OK** to return to the Edit LabVIEW VI Call dialog box.

    Notice that the Value column for the `Input Cluster` parameter now shows `Locals.ContainerData`.
14. Type `Step.Result.ReportText` in the Value column for the `ReportText` output parameter.
15. Click **OK** to close the Edit LabVIEW VI Call dialog box and return to the Sequence File window.

    When TestStand calls the VI, it will pass the values in the `ContainerData` local variable to the **Input Cluster** control on the VI and output the string element of the `Input Cluster` parameter to the `ReportText` property of the step.
16. Select **Execute»Single Pass** to start a new execution of the sequence using the Single Pass Execution entry point.

    When the execution is complete, the resulting report shows the text returned from `VI with Cluster Input.vi`.

You have completed this tutorial. In the next chapter, you will learn how to configure the LabVIEW Adapter.

5

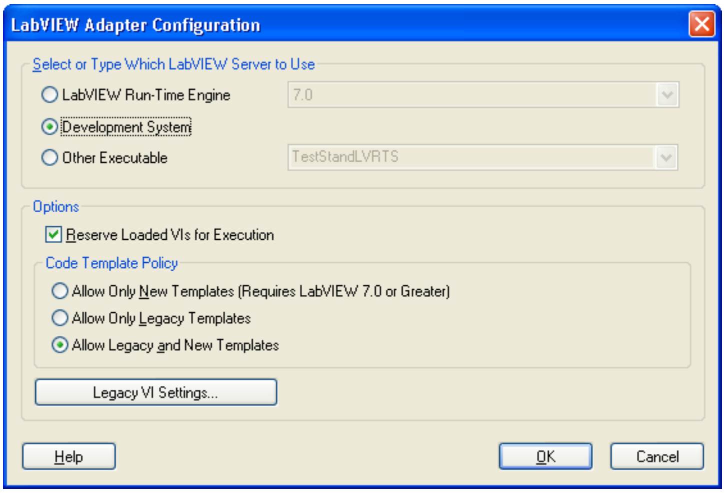
Configuring the LabVIEW Adapter

In this chapter, you will learn how to configure the various settings of the LabVIEW Adapter. To access the LabVIEW Adapter Configuration dialog box, you must first launch the general Adapter Configuration dialog box by selecting **Adapters** from the **Configure** menu. Then, select **LabVIEW** from the Adapter column and click **Configure**.

Selecting a LabVIEW Server

The TestStand LabVIEW Adapter can run VIs using any of the following LabVIEW environments, or *servers*: the LabVIEW development system, the LabVIEW Run-Time Engine, or a LabVIEW executable built with an ActiveX server enabled.

Use the LabVIEW Adapter Configuration dialog box, which is shown in Figure 5-1, to select the server you want TestStand to use.


LabVIEW Adapter Configuration

Select or Type Which LabVIEW Server to Use
- ○ LabVIEW Run-Time Engine — 7.0
- ● Development System
- ○ Other Executable — TestStandLVRTS

Options
- ✓ Reserve Loaded VIs for Execution

Code Template Policy
- ○ Allow Only New Templates (Requires LabVIEW 7.0 or Greater)
- ○ Allow Only Legacy Templates
- ● Allow Legacy and New Templates

Figure 5-1. LabVIEW Adapter Configuration Dialog Box

- **LabVIEW Run-Time Engine**—Provides optimal performance when calling LabVIEW VIs. This option does not allow you to create or edit VIs from TestStand or debug VIs that TestStand calls.

  You must have the LabVIEW Run-Time Engine 7.0 or later on the same computer as TestStand in order to use this option.
- **Development System**—Allows you to create or edit VIs from TestStand and debug VIs that TestStand calls in LabVIEW.

  You must have the LabVIEW development system installed on the same computer as TestStand to use this option.
- **Other Executable**—Uses a LabVIEW executable that you build with LabVIEW's Build Application or Shared Library (DLL) functionality. This option does not allow you to create or edit VIs from TestStand or debug VIs that TestStand calls.

  To use this option, enter the name of the ActiveX server associated with your LabVIEW executable.

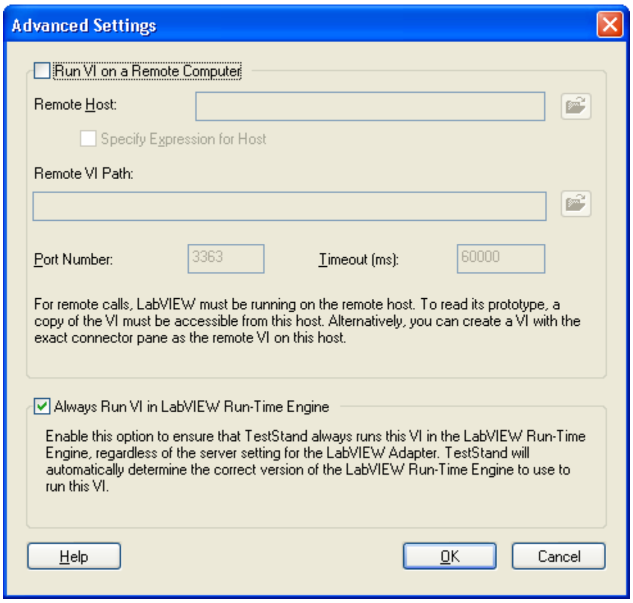
Tip The name of the ActiveX server is the same name that you entered in the LabVIEW Application Builder as the Server Name on the Application tab of the Build Application or Shared Library (DLL) dialog box in LabVIEW.

The executable must be installed and registered on the same computer as TestStand to use this option. To register your executable as an ActiveX server, launch the executable once. You can find an example server VI and application build script in the following location: `<TestStand>\Components\NI\RuntimeServers\LabVIEW`.

Note If you select **LabVIEW Run-Time Engine** or **Other Executable** as your TestStand server, you must fully deploy your VIs before calling them from TestStand. The LabVIEW Run-Time Engine or built executable must be able to find the complete hierarchies of your VIs, including any subVIs from the LabVIEW `vi.lib` directory. Refer to Chapter 14, *Deploying TestStand Systems*, in the *TestStand Reference Manual* for more information about deploying your VIs for use with TestStand.

Per-Step Configuration of the LabVIEW Server

You can direct TestStand to always use the LabVIEW Run-Time Engine to execute a step. Make this selection in the Advanced Settings dialog box, which you launch by clicking the **Advanced** button on the Edit LabVIEW VI Call dialog box. The Advanced Settings dialog box is illustrated in Figure 5-2.



[ ] Run VI on a Remote Computer

Remote Host:

[ ] Specify Expression for Host

Remote VI Path:

Port Number: 3363

Timeout (ms): 60000

For remote calls, LabVIEW must be running on the remote host. To read its prototype, a copy of the VI must be accessible from this host. Alternatively, you can create a VI with the exact connector pane as the remote VI on this host.

[✓] Always Run VI in LabVIEW Run-Time Engine

Enable this option to ensure that TestStand always runs this VI in the LabVIEW Run-Time Engine, regardless of the server setting for the LabVIEW Adapter. TestStand will automatically determine the correct version of the LabVIEW Run-Time Engine to use to run this VI.

Figure 5-2. Advanced Settings Dialog Box

When you enable the Always Run VI in LabVIEW Run-Time Engine option, TestStand selects the appropriate version of the LabVIEW Run-Time Engine according to the version of LabVIEW in which the VI was last compiled.

Note The Always Run VI in LabVIEW Run-Time Engine option overrides the global LabVIEW Server Selection option in the LabVIEW Adapter Configuration dialog box. Use this option when you create tools and step types that use the LabVIEW Adapter that you do not want to be affected by the global LabVIEW Server Selection option.

Reserve Loaded VIs for Execution

Enable the **Reserve Loaded VIs for Execution** option in the LabVIEW Adapter Configuration dialog box to instruct TestStand to reserve any VIs that TestStand loads for calling with the LabVIEW Adapter.

This option reduces the amount of time required for TestStand to call VIs. Additionally, enabling the Reserve Loaded VIs for Execution option makes references that you create in a VI you call from TestStand—such as I/O, ActiveX, and synchronization references—persist across calls to other VIs. You can store these references in a TestStand property and pass them to subsequent VIs that you call from TestStand.

While reserving VIs with this option reduces the amount of time required for TestStand to call the VIs, this option also blocks other applications from using any VIs loaded by TestStand, including subVIs of the VIs that TestStand calls directly.

If you open a reserved VI in LabVIEW, you will notice that the Run arrow becomes visible.

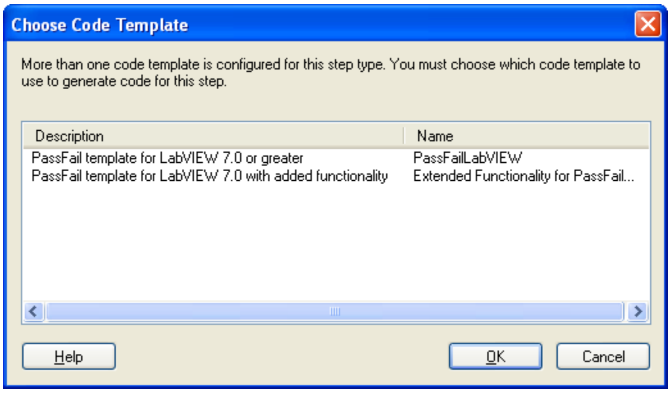
This arrow indicates that the VI is reserved and cannot be edited. To edit a VI that TestStand has reserved, select **Edit Code** from the step's context menu to open the VI in TestStand. You can also select **File»Unload All Modules** from the Sequence Editor before opening the VI in LabVIEW.

You must also ensure that you close any references you create in your VIs once you are finished with them. If TestStand reserves VIs when it loads them, LabVIEW will not automatically close the references until TestStand unloads the VIs that created the references. Failing to close the references could result in a memory leak in your test system.

Code Template Policy

The Code Template Policy section of the LabVIEW Adapter Configuration dialog box allows you to specify whether TestStand allows you to create new test VIs using old, or *legacy*, VI templates. These legacy VIs are callable from previous versions of TestStand and the LabVIEW Test Executive. Refer to Appendix C, *Calling Legacy VIs*, for more information about legacy TestStand VIs.

If you have configured the LabVIEW Adapter using the Allow Only New Templates (Requires LabVIEW 7.0 or Greater) option and then create a new VI from the Edit LabVIEW VI Call dialog box, TestStand either immediately creates a new VI based on the code template for the specified step type or, if the step type has multiple code templates available, launches the Choose Code Template dialog box illustrated in Figure 5-3, from which you select the code template to use for the new VI.

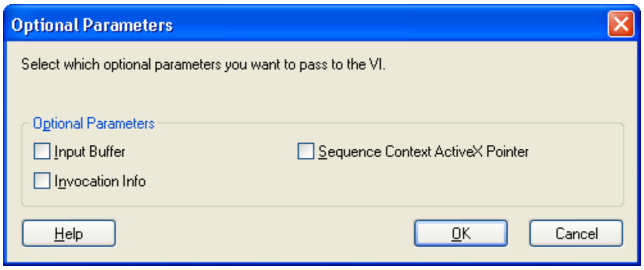


More than one code template is configured for this step type. You must choose which code template to use to generate code for this step.

Description	Name
PassFail template for LabVIEW 7.0 or greater	PassFailLabVIEW
PassFail template for LabVIEW 7.0 with added functionality	Extended Functionality for PassFail...

Figure 5-3. Choose Code Template Dialog Box

If you have configured the LabVIEW Adapter using the Allow Only Legacy Templates option, TestStand launches the Optional Parameters dialog box, in which you choose optional parameters such as Input Buffer, Sequence Context ActiveX Pointer, or Invocation Info that you want to include as input parameters for the VI, as shown in Figure 5-4.



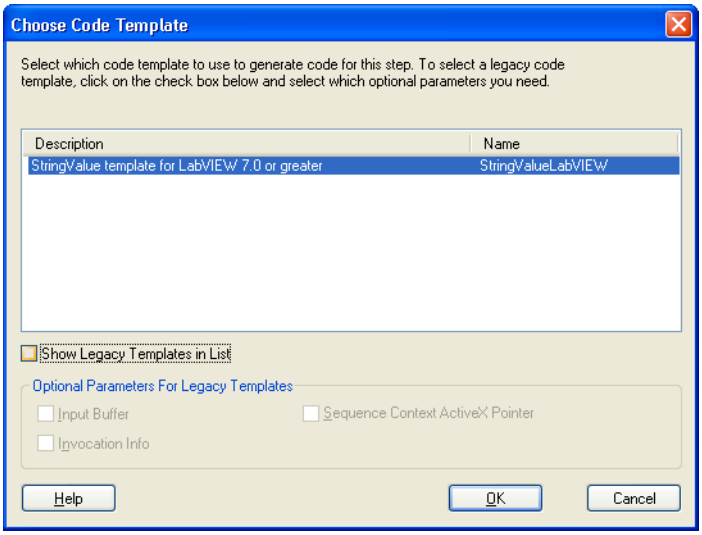
Select which optional parameters you want to pass to the VI.

Optional Parameters

- [ ] Input Buffer
- [ ] Sequence Context ActiveX Pointer
- [ ] Invocation Info

Figure 5-4. Optional Parameters Dialog Box

If you have configured the LabVIEW Adapter using the Allow Legacy and New Templates option, TestStand launches the Choose Code Template dialog box. You can choose a new template from the list of available templates for the step type, or enable the **Show Legacy Template** option to choose the optional parameters that you want to use as input parameters for the VI. This dialog box is illustrated in Figure 5-5.



Select which code template to use to generate code for this step. To select a legacy code template, click on the check box below and select which optional parameters you need.

Description	Name
StringValue template for LabVIEW 7.0 or greater	StringValueLabVIEW

Show Legacy Templates in List
Optional Parameters For Legacy Templates
Input Buffer
Sequence Context ActiveX Pointer
Invocation Info
Help
OK
Cancel

Figure 5-5. Choose Code Template Dialog Box

Legacy VI Settings

From within the LabVIEW Adapter Configuration dialog box, click **Legacy VI Settings** to launch the Legacy VI Settings dialog box, in which you can configure settings relevant to calling legacy test VIs. The Legacy VI Settings dialog box contains expressions that the LabVIEW Adapter evaluates to generate values to pass to the VI in the various **Invocation Info** cluster fields.

Legacy VIs can use the **Invocation Info** cluster as an optional input. Refer to Appendix C, *Calling Legacy VIs*, for more information about the **Invocation Info** cluster.

6 Creating Custom User Interfaces in LabVIEW

This chapter discusses the tools that TestStand provides for creating custom operator interfaces and for creating user interfaces for other components, such as custom step types.

Tip National Instruments recommends that you read Chapter 9, *Creating Custom Operator Interfaces*, of the *TestStand Reference Manual*, before proceeding with this chapter. For additional information, refer to the *TestStand User Interface Controls Reference Poster* for an illustrated overview of the TestStand User Interface Controls.

Note Unless otherwise noted, all controls and VIs described in this chapter require LabVIEW 7.0 or later.

Note The TestStand User Interface Controls are not supported in Windows 98.

TestStand User Interface Controls

The TestStand User Interface (UI) Controls are located in the **Controls»All Controls»TestStand** palette in LabVIEW, which is illustrated in Figure 6-1.

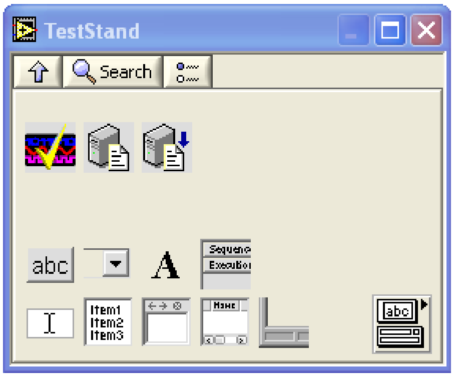


Figure 6-1. User Interface Controls Palette

When you place these controls on the front panel of a VI, you can program them using the LabVIEW ActiveX functionality.

You can also configure the controls interactively using the LabVIEW Property Browser or any property pages featured by the controls. To open the LabVIEW Property Browser, right-click any control and select **Property Browser**. To open property pages, right-click any eligible control and select **<Control Name>»Properties**.

For general information about programming the TestStand API from LabVIEW, refer to Appendix B, *Using the TestStand ActiveX APIs in LabVIEW*.

TestStand Utility Library

The TestStand Utility Library VIs are the LabVIEW versions of the functions in the TestStand Utility Functions Library, or TSUtil. These VIs are located in the **Functions»All Functions»TestStand** palette, which is illustrated in Figure 6-2.

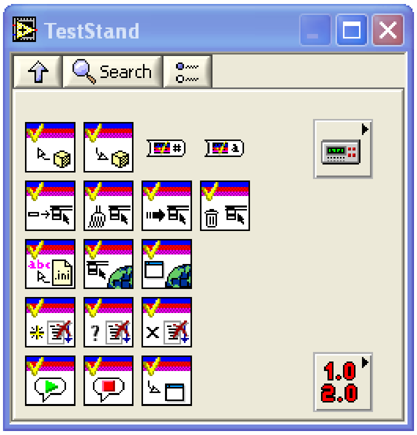


Figure 6-2. TestStand Utility Library Palette

The TestStand Utility Library VIs assist user interface developers with the following tasks:

- Inserting menu items that automatically execute commands provided by the TestStand UI Controls.
- Localizing the strings on your user interface. These VIs are compatible with LabVIEW 6.1 or later.

- Making dialog boxes that are launched by LabVIEW VIs modal to TestStand applications. These VIs are compatible with LabVIEW 6.1 or later.
- Checking whether an execution that calls a VI has stopped. These VIs are compatible with LabVIEW 6.1 or later.
- Setting and getting the values of TestStand properties and variables.

To access the help for a TestStand Utility Library VI, drop the VI onto a LabVIEW diagram, right-click it and then select **Help** from the context menu.

Creating Custom Operator Interfaces

Operator interfaces that use the TestStand UI Controls typically perform the following basic operations:

- Configure connections, commands, and other control settings
- Register to handle events sent by the controls
- Start TestStand
- Wait in a main event loop until you close the application
- Shut down TestStand

Operator interfaces may also have a menu bar containing items that invoke TestStand commands, as well as non-TestStand items.

For additional information about creating a TestStand operator interface using the TestStand UI Controls in LabVIEW, refer to the example operator interfaces included with TestStand. Begin with the simple operator interface example, `<TestStand>\OperatorInterfaces\NI\Simple\LabVIEW\TestExec.llb\Simple OI - Top-Level VI.vi`. For a more advanced example that includes menus and localization, refer to the full-featured example, `<TestStand>\OperatorInterfaces\NI\Full-Featured\LabVIEW\TestExec.llb\Full OI - Top Level VI.vi`.

To customize the example operator interfaces, copy the operator interface directory and its contents from the `NI` subdirectory to the `<TestStand>\OperatorInterfaces\User` subdirectory before beginning your customizations. This ensures that newer installations of TestStand will not overwrite your custom operator interface.

Note Example operator interfaces that use the TestStand API instead of the TestStand UI Controls are available in the `<TestStand>\OperatorInterfaces\NI\TestStand 2.0.1 Operator Interfaces (Old)` directory. These operator interfaces are compatible with LabVIEW 6.1 or later.

Configuring the TestStand UI Controls

Refer to the following example operator interface VIs for examples of configuring connections, commands, and other settings for the TestStand UI Controls:

- `Simple OI - Configure Application Manager.vi`
- `Simple OI - Configure SequenceFileView Manager.vi`
- `Simple OI - Configure ExecutionView Manager.vi`
- `Full OI - Configure StatusBar.vi`
- `Full OI - Configure SequenceFileView Manager.vi`
- `Full OI - Configure ListBar.vi`
- `Full OI - Configure ExecutionView Manager.vi`

Handling Events

TestStand UI Controls generate events to notify your application of user input and application events, such as the completion of an execution. To handle an event in LabVIEW, you register a callback VI, which is automatically called when the control generates the event. Follow these steps to perform this registration using the Register Event Callback function, which is located in the ActiveX subpalette of the LabVIEW Full or Professional Development System:

1. Wire the reference to the control that sends the event you want to handle to the **Event** input of the Register Event Callback function.
2. Click the **Event** input terminal and select the specific event you want to handle from the list.
3. If you want to pass custom data to the callback VI, wire the custom data to the **User Parameter** input of the Register Event Callback function. The User Parameter input can be any data type.
4. Right-click the **VI Ref** input of the Register Event Callback function and select **Create Callback VI**.

   LabVIEW creates an empty callback VI with the correct input parameters for the particular event, including an input parameter for any custom data that you wired to the User Parameter input in Step 3.

5. Save the new callback VI.

   The block diagram containing the Register Event Callback function now shows a Static VI Reference node wired to the **VI Ref** input of the function. This node returns a strictly-typed reference to the new callback VI.

6. Complete the diagram of the callback VI so that it performs the operation you specify when the control generates the event.

Note When your application is finished handling events for the control, it must close the event callback refnum output of the Register Event Callback function using the Unregister for Events function, which is located in the **Application Control»Events** palette.

Figure 6-3 illustrates registering a callback VI to handle the Break event for the TestStand Application Manager control.

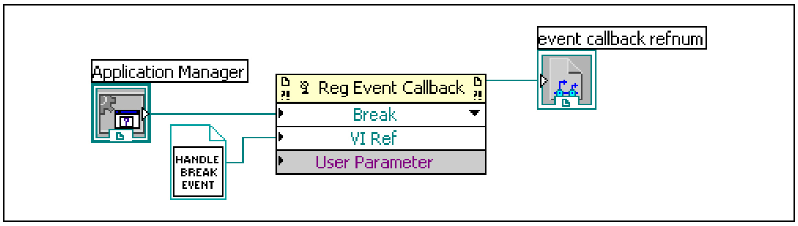


Figure 6-3. Register For Break Event

You can register to handle multiple events using the same Register Event Callback function by resizing the node to show multiple sets of input terminals. Refer to the following example operator interface VIs for examples of registering to handle events from the TestStand UI Controls:

- `Simple OI - Configure Event Callbacks.vi`
- `Full OI - Configure Event Callbacks.vi`

Starting TestStand

Start TestStand by invoking the Application Manager control Start method. Refer to the following example operator interfaces for examples of using this method:

- `Simple OI - Top-Level VI.vi`
- `Full OI - Top-Level VI.vi`

The Main Event Loop and Shutting Down TestStand

Operator interface applications wait in a main event loop after starting TestStand. This main event loop can handle many events, such as menu selections and LabVIEW control value changes. However, the main event loop must at least handle the events that will stop the operator interface application.

To stop an operator interface application, click the **Close** box or execute the **Exit** command through either a TestStand menu or a Button control.

When you click the Close box, the Event structure in the main event loop handles the <VI Name>:Panel Close? event. The block diagram that handles the event then invokes the Application Manager control Shutdown method and discards the event. If the Shutdown method returns `True`, indicating that TestStand is ready to shut down, the main event loop stops. If the Shutdown method returns `False`, TestStand cannot shut down until the executions complete or sequence files are unloaded. In this case, the main event loop continues to wait until TestStand can shut down. When TestStand is ready, the Application Manager control sends the ExitApplication event.

The callback VI for the Application Manager control ExitApplication event generates a LabVIEW User event called the Quit Application event. This event, which is created and handled in the example operator interface VIs, informs the main event loop that it can stop.

Refer to the following example operator interface VIs for examples of the main event loop and shutting down TestStand. These VIs also provide examples of creating, generating, and handling the Quit Application event.

- `Simple OI - Top-Level VI.vi`
- `Simple OI - ExitApplication Event Callback.vi`
- `Full OI - Top-Level VI.vi`
- `Full OI - Create Quit Application Event.vi`
- `Full OI - ExitApplication Event Callback.vi`

Menu Bars

The TestStand Utility Library provides the following VIs for creating and handling menu items that execute TestStand UI Control commands:

- `TestStand - Insert Commands in Menu.vi`
- `TestStand - Cleanup Menus.vi`
- `TestStand - Remove Commands From Menus.vi`
- `TestStand - Execute Menu Command.vi`

Since maintaining the current state of the menu bar can be very difficult, you should only handle the menu bar when it is required. The Event structure in the main event loop handles the <VI Name>:Menu Activation? event to determine when you open a menu or select a shortcut key that may be linked to a menu item. The block diagram that handles this event can then rebuild the menu bar.

To handle when the user makes a menu selection, the Event structure in the main event loop handles the <VI Name>:Menu Selection (User) event. The diagram that handles the event calls `TestStand - Execute Menu Command.vi` to execute the appropriate TestStand command. If the item is not a TestStand menu item, the diagram that handles the event manually handles the menu selection.

Refer to the following example operator interface VIs for examples of rebuilding the menu bar and handling menu selections:

- `Full OI - Top-Level VI.vi`
- `Full OI - Rebuild Menu Bar.vi`

Localization

The TestStand UI Controls and TestStand Utility Library provide tools that localize your operator interfaces based on the TestStand language setting. Use the following Utility Library VIs to localize your operator interface:

- `TestStand - Get Resource String.vi`
- `TestStand - Localize Menu.vi`
- `TestStand - Localize Front Panel.vi`

Refer to the following example operator interface VIs for examples of localizing operator interface panels:

- `Full OI - Localize Operator Interface.vi`
- `Full OI - About Box.vi`

Other User Interface Utilities

Making a Dialog VI Modal to TestStand

VIs that TestStand calls may display dialog boxes that are modal to TestStand application windows such as the TestStand Sequence Editor or a TestStand Operator Interface.

The TestStand Utility Library provides two VIs that make a dialog box modal to TestStand application windows: `TestStand - Start Modal Dialog.vi` and `TestStand - End Modal Dialog.vi`.

The following example demonstrates this technique: `<TestStand>\Examples\ModalDialogs\LabVIEW`.

Checking For Stopped Execution

VIs called by TestStand may display dialog boxes or perform other lengthy operations. Therefore, it can be useful to have those VIs periodically check whether their parent execution has been terminated or aborted. This allows the VIs to stop gracefully and allows their parent execution to terminate or abort.

The TestStand Utility Library provides the following VIs that enable VIs called by TestStand to verify whether their calling execution has been stopped:

- `TestStand - Initialize Termination Monitor.vi`
- `TestStand - Check Termination Monitor Status.vi`
- `TestStand - Close Termination Monitor.vi`

You can also refer to the dialog box VIs in the following examples for a demonstration of this technique:

- `<TestStand>\Examples\Demo\LabVIEW\Computer Motherboard Test`
- `<TestStand>\Examples\Demo\LabVIEW\Auto`

A Calling LabVIEW VIs on Remote Systems

TestStand allows you to directly call LabVIEW VIs on remote computers. These computers can be machines running the LabVIEW development system or a LabVIEW executable, or they can be PXI controllers running the LabVIEW Real-Time (RT) module. Because TestStand uses the LabVIEW VI Server to run VIs remotely, these computers can use any platform that LabVIEW supports, including Linux, Solaris, and Mac OS.

To call a VI remotely, you must configure the TestStand step to specify that the call occurs on a remote computer. In addition, you must configure the remote computer to allow TestStand to call VIs that are located on that computer.

Note You must configure the computer that is running TestStand so that it has network access to the remote computer running the LabVIEW VI Server.

Configuring a Step to Run Remotely

1. Click **Advanced** on the Edit LabVIEW VI Call dialog box to launch the Advanced Settings dialog box.

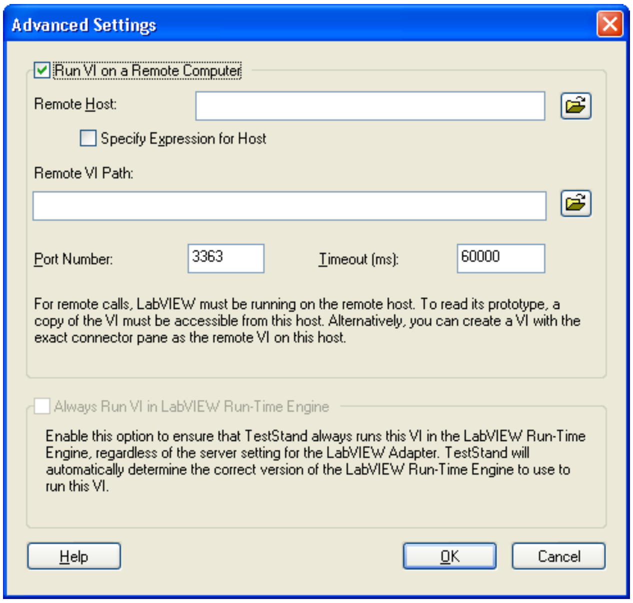
   Use the Advanced Settings dialog box to specify the name, or an expression that evaluates to the name, of the remote computer on which you want to run the VI.

Note The VI must be present on the local computer in order for TestStand to be able to read the connector pane information from the VI.

2. If the remote computer is running the LabVIEW development system or a LabVIEW executable, specify the path to where the VI is located on the remote computer in the Remote VI Path text box, which is shown in Figure A-1.

   If your remote computer is a PXI controller running LabVIEW RT, TestStand will either download the VI to the remote computer or load the VI using the Remote VI Path that you specified in the Advanced Settings dialog box.

Note TestStand skips this step if the VI is already present in memory on the controller at the time TestStand loads the code module for the step.


Advanced Settings
Run VI on a Remote Computer
Remote Host:
Specify Expression for Host
Remote VI Path:
Port Number:
Timeout (ms):
Always Run VI in LabVIEW Run-Time Engine
Help
OK
Cancel

[✓] Run VI on a Remote Computer

Port Number: 3363

Timeout (ms): 60000

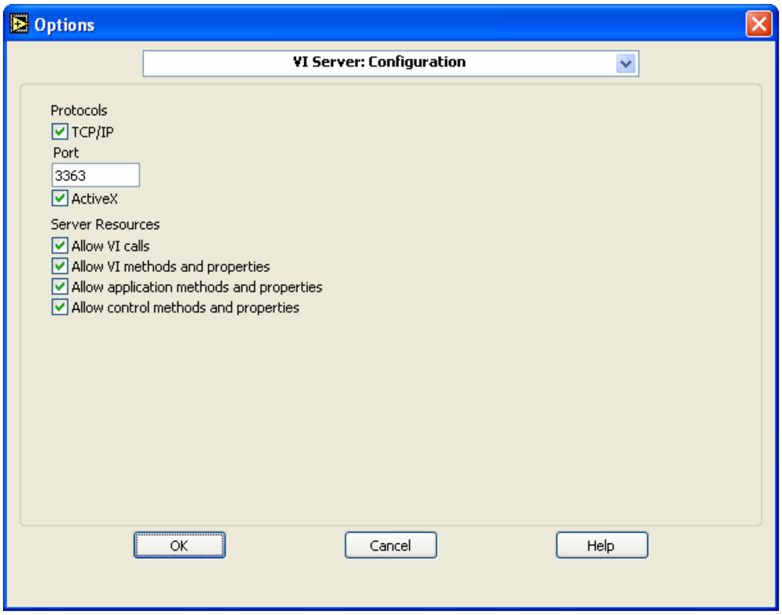
For remote calls, LabVIEW must be running on the remote host. To read its prototype, a copy of the VI must be accessible from this host. Alternatively, you can create a VI with the exact connector pane as the remote VI on this host.

Enable this option to ensure that TestStand always runs this VI in the LabVIEW Run-Time Engine, regardless of the server setting for the LabVIEW Adapter. TestStand will automatically determine the correct version of the LabVIEW Run-Time Engine to use to run this VI.

Figure A-1. Advanced Settings Dialog Box

Configuring a LabVIEW VI Server to Run VIs Remotely

The LabVIEW development system or built executable must be running and configured on the remote machine in order to allow VI calls through the TCP/IP protocol of the VI Server. Configure these settings on the VI Server: Configuration property page of the Options dialog box in LabVIEW (**Tools»Options**), which is shown in Figure A-2.



Protocols

☑ TCP/IP

Port

3363

☑ ActiveX

Server Resources

☑ Allow VI calls

☑ Allow VI methods and properties

☑ Allow application methods and properties

☑ Allow control methods and properties

Figure A-2. VI Server: Configuration Property Page of the Options Dialog Box

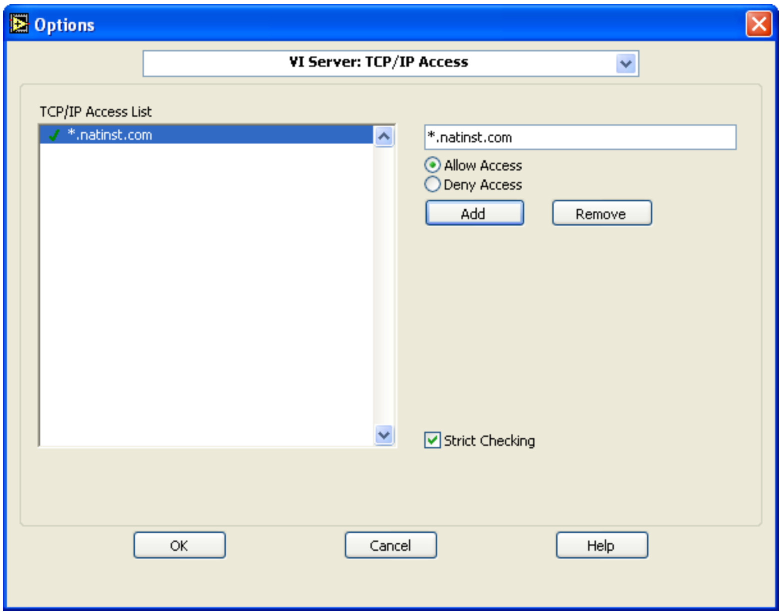
1. In the **VI Server: Configuration** property page of the Options dialog box, enable the **TCP/IP** protocol and the **Allow VI calls** options.

   You can also use this dialog box to specify the TCP/IP port that the server uses.

Note The port specified in LabVIEW must be the same port that you specified in the Advanced Settings dialog box, which you access from the Edit LabVIEW VI Call dialog box.

2. Configure any computers that you want to allow access to the LabVIEW VI Server in the **VI Server: TCP/IP Access** property page of the Options dialog box, which is shown in Figure A-3.

   You can also use this dialog box to specify specific computers or entire domains that can call VIs on the server machine.



TCP/IP Access List: ✓ *.natinst.com

*.natinst.com

[✓ Allow Access] Deny Access

✓ Strict Checking

Figure A-3. VI Server: TCP/IP Access Property Page of the Options Dialog Box

3. Configure the VIs that you want to allow to be called through the LabVIEW VI Server in the **VI Server: Exported VIs** property page of the Options dialog box, which is shown in Figure A-4.

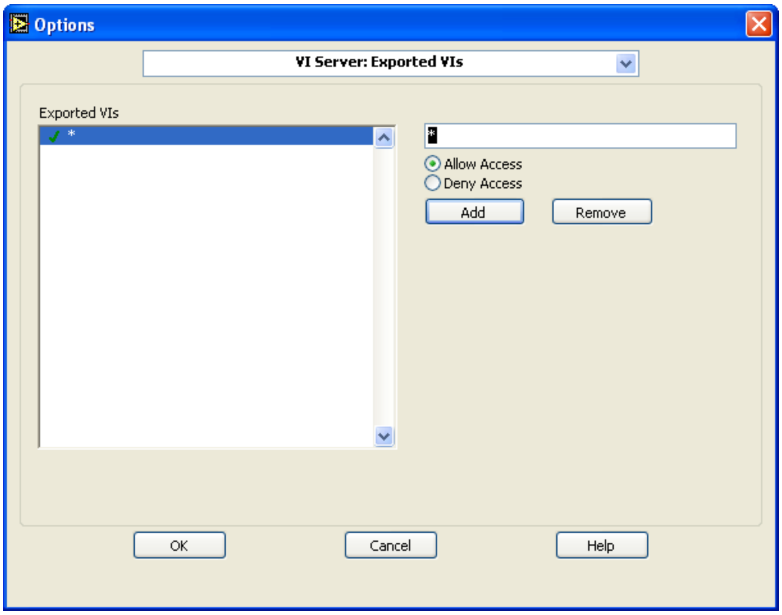


Figure A-4. VI Server: Exported VIs Property Page of the Options Dialog Box

Note All VIs that you want to call remotely from TestStand must be exported. The default setting in LabVIEW is to export all VIs, and is indicated by an asterisk (*).

Configuring a LabVIEW RT Server to Run VIs

To perform the configuration described in the *Configuring a LabVIEW VI Server to Run VIs Remotely* section, launch LabVIEW on the host computer. Next, select the appropriate RT target computer and select **Tools»RT Target <IP Address/Host Name> Options**.

In addition, you must also configure the RT target computer to allow access to the computer running TestStand if you want TestStand to download VIs to the RT target computer.

Perform this configuration on the **RT Target: Access** property page of the Options dialog box in LabVIEW, which is illustrated in Figure A-5.

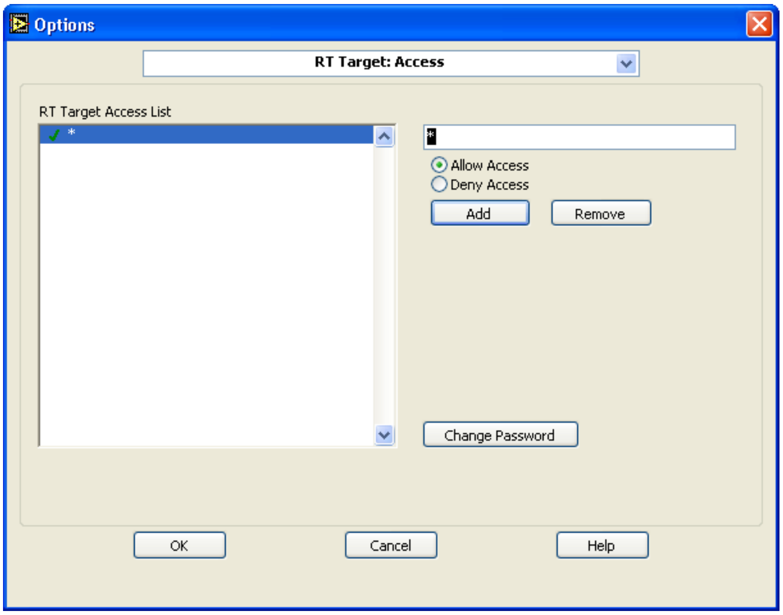


Figure A-5. RT Target: Access Property Page of the Options Dialog Box

Note When you have finished configuring your target computer, be sure to untarget the PXI controller.

B

Using the TestStand ActiveX APIs in LabVIEW

In some cases you will need to program the TestStand API or TestStand UI Controls from your LabVIEW test and user interface VIs. This chapter contains information about programming with the TestStand API and TestStand UI Controls from LabVIEW.

Chapter 19, *Windows Connectivity*, in the *LabVIEW User Manual* contains fundamental information about ActiveX concepts and how to use LabVIEW as an ActiveX client. National Instruments recommends that you become familiar with this material before proceeding with this chapter.

Invoking Methods

TestStand objects have methods that you invoke to perform an operation or function on them. In LabVIEW, you invoke methods using the **Invoke** node, which you can access from the ActiveX palette. The block diagram in Figure B-1 illustrates invoking the UnloadModules method of a Sequence object to unload all code modules in the sequence.

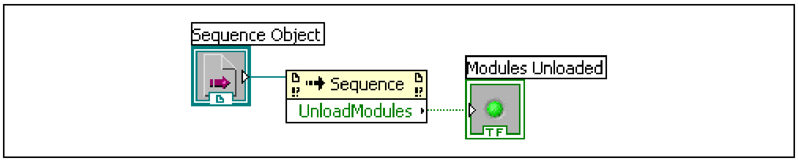


Figure B-1. Invoking the UnloadModules Method

Accessing Built-In Properties

TestStand defines a number of built-in properties that are always present for objects such as steps and sequences. Nearly every kind of object in TestStand has built-in properties, which are static with respect to the TestStand API. This means that the TestStand API has knowledge about each of these properties, which it uses to allow you to access these properties in the programming language you specify. Examples of built-in properties are the Name property of the Sequence object and the Sequence property of the SequenceContext object.

In LabVIEW, you access built-in properties using the **Property** node, which is available on the ActiveX palette.

The block diagram in Figure B-2 illustrates obtaining the value of the Name property from a Sequence object.

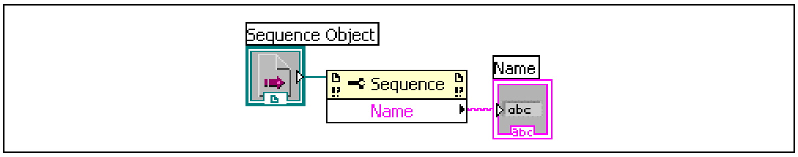


Figure B-2. Obtaining the Value of the Name Property from a Sequence Object

The block diagram in Figure B-3 illustrates obtaining a reference to a step of a sequence referenced by a Sequence object.

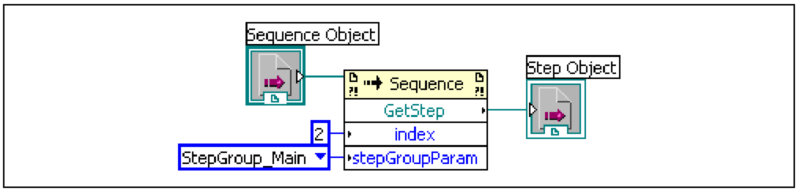


Figure B-3. Obtaining a Reference to a Step of a Sequence Referenced by a Sequence Object

Accessing Dynamic Properties

TestStand allows you to define your own custom step properties, sequence local variables, sequence file global variables, and station global variables. Because the TestStand API has no knowledge of the variables and custom step properties that you define, these variables and properties are dynamic with respect to the TestStand API. The TestStand API provides the PropertyObject class so that you can access dynamic properties and variables from within code modules. Instead of using defined constants, use lookup strings to identify specific properties by name.

To access dynamic properties of an object, you must first convert the specific object reference to a PropertyObject reference using the object's AsPropertyObject method. Then, use the PropertyObject interface to access custom properties of the object by using a lookup string to specify the specific custom property.

The block diagram in Figure B-4 illustrates using the GetValString method on the PropertyObject interface of a Step object to obtain the error message value for the current step.

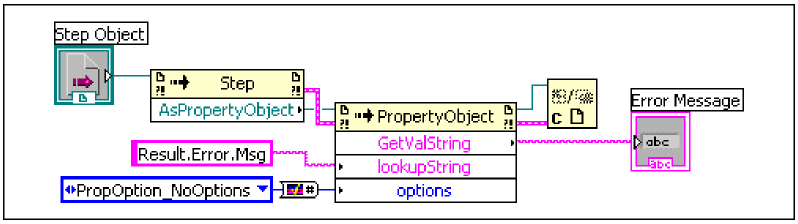


Figure B-4. Using the GetValString Method to Obtain the Error Message Value for the Current Step

You can easily access dynamic properties of a SequenceContext object by using one of the following polymorphic VIs which are included in TestStand: `TestStand - Get Property Value.vi` and `TestStand - Set Property Value.vi`. These VIs are located in the TestStand subpalette of the LabVIEW Functions palette.

The block diagram in Figure B-5 illustrates obtaining the error message value for the current step using `TestStand - Get Property Value.vi`.

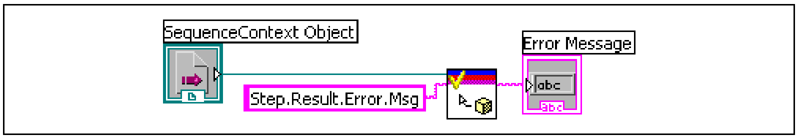


Figure B-5. Obtaining the Error Message for the Current Step

Releasing ActiveX References

When a method or property returns an ActiveX reference, you must release it using the Automation Close function in LabVIEW, which is located in the ActiveX palette.

Note If you fail to release the ActiveX reference, LabVIEW will not release it for you until your VI hierarchy finishes executing. Repeatedly opening large numbers of references without closing them can cause your system to run out of memory.

Using TestStand API Constants and Enumerations

Some TestStand API methods require string and numeric constant input arguments. The allowed values of these arguments are organized into groups that correspond to different properties and methods. For example, the PropertyObject.SetValNumber method has an options input argument that accepts many different numeric constants.

It can be difficult to remember all of the available string and numeric constants for the TestStand API properties and methods. To facilitate programming with the TestStand API within LabVIEW VIs, TestStand provides two enumerated constant VIs—`TestStand API String Constants.vi` and `TestStand API Numeric Constants.vi`—which are located in the TestStand subpalette of the LabVIEW Functions palette.

Use the TestStand API String Constants VI to quickly locate and select the string constant arguments for a property or method in the API. Use the TestStand API Numeric Constants VI to locate and select the various numeric constant arguments that you can use with TestStand API properties and methods. Use both of these VIs in conjunction with the information in the *TestStand Help* regarding constants associated with the TestStand API methods and properties.

You can use the OR function in LabVIEW to combine more than one of the numeric constants. If you need to combine more than two of the constants, use the Compound Arithmetic function with its mode set to OR.

The block diagram in Figure B-4 shows how to use the TestStand API Numeric Constants VI to obtain the value of the PropOptions_NoOptions constant.

In addition, some methods in the TestStand API require enum input arguments, For these methods, right-click the input parameter on the **Invoke** node in LabVIEW and select **Create»Constant** to create a LabVIEW ring constant. Then, select the desired value in the resulting constant.

Getting a Different Interface for a TestStand Object

In some cases, you may need to obtain a different interface for a TestStand object than the one you currently have. In ActiveX/COM terminology, this action is known as a QueryInterface.

For example, if you have a Module reference to a LabVIEWModule object and need to access its LabVIEWModule interface, perform a QueryInterface on the Module object to obtain that interface. In LabVIEW, use the **Variant To Data** function, which is located on the ActiveX palette, on the reference to accomplish this task.

The block diagram in Figure B-6 shows how to obtain the LabVIEWModule interface of a Module object to get the VIDescription property of the object. Note that you must release the reference returned by the Variant To Data function when you are finished with it.

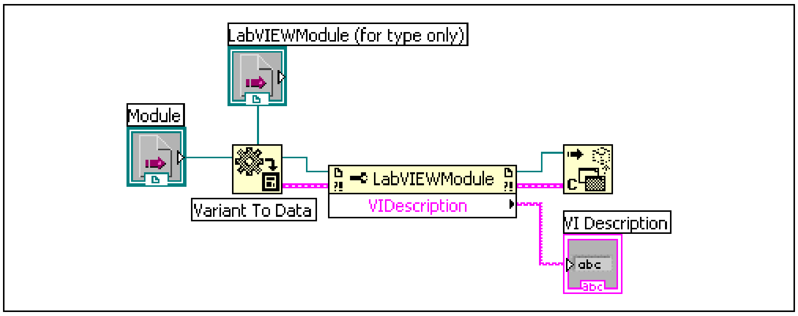


Figure B-6. Converting a Module Reference to the LabVIEWModule Type

Acquiring a Derived Class from the PropertyObject Class

In some cases, you may need to obtain a reference to a TestStand object using the PropertyObject class methods. You may then want to access one of the static properties of that TestStand object, such as obtaining the run mode for the third step in the Main step group of the currently executing sequence. For methods in the PropertyObject class that can return objects derived from PropertyObject, you must acquire the derived interface for the object to access the built-in properties and methods of the derived class. Acquire the derived interface for an object using the method described in the *Getting a Different Interface for a TestStand Object* section.

The block diagram in Figure B-7 illustrates obtaining a reference to a Step object from a SequenceContext object using a lookup string.

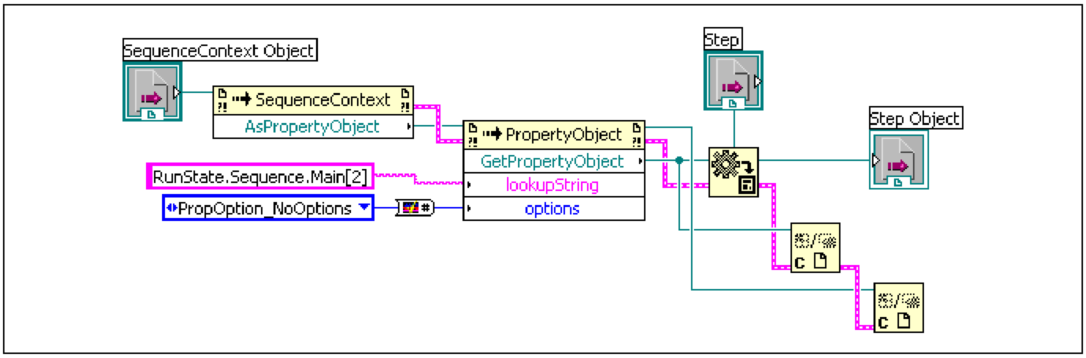


Figure B-7. Obtaining a Reference to a Step Object from a SequenceContext Object Using a Lookup String

Setting the Preferred Execution System for LabVIEW VIs

If your VI calls synchronous methods of the TestStand API, you may need to correctly set the LabVIEW Preferred Execution System for your VIs. If you call synchronous methods that will not return until the LabVIEW server executes a VI on behalf of TestStand, the VI that calls these methods and the VI that TestStand is attempting to run using the LabVIEW VI Server cannot be set to run in the same LabVIEW execution system. If the VIs are set to run in the same execution system, you will experience a deadlock since the execution system that the VI needs to run in will be consumed by the execution of the synchronous TestStand method. In addition, since LabVIEW handles ActiveX communication through its user interface execution system, neither of the VIs in this scenario may be set to run in the user interface execution system.

For example, you can have a LabVIEW code module that calls the Engine.NewExecution method followed by the Engine.WaitForEnd method, and a new execution that calls LabVIEW code modules. Deadlock can occur if either of the VIs in this scenario use Same As Caller or User Interface as its preferred execution system. In addition, both VIs in this scenario must use different preferred execution system settings. The LabVIEW execution system is configured in the VI Properties dialog box for each individual VI.

C

Calling Legacy VIs

Prior to version 3.0, TestStand could only call VIs with a specific set of controls and indicators. Using TestStand 3.0 and LabVIEW 7.0, you can now call VIs with a wide variety of connector panes, including VIs with legacy configurations.

If you are using LabVIEW 6.1, you can only call legacy VIs.

Format of Legacy VIs

The **Test Data** cluster and **error out** cluster indicators are required outputs for all legacy-style VIs. The **Input Buffer**, **Invocation Info**, and **Sequence Context** controls are optional inputs to legacy VIs. Legacy VIs can contain any combination of these controls.

TestStand does not require a particular connector pane pattern or require that the controls and indicators be assigned to specific terminals. TestStand only requires that you assign the controls and indicators to some terminal on the connector pane of the VI.

While you would usually create new VIs from the Edit LabVIEW VI Call dialog box for steps that use the LabVIEW Adapter, TestStand can also create legacy-style VIs. Chapter 5, *Configuring the LabVIEW Adapter*, details how to configure the LabVIEW Adapter for creating new legacy-style VIs.

The following sections discuss each of the required and optional VI controls.

Note The specific control and indicator labels described in this section are required. Do not modify them in any way.

Test Data Cluster

The LabVIEW Adapter must use the **Test Data** cluster to return result data from the VI to TestStand. TestStand can then use the data to make a PASS/FAIL determination.

Figure C-1 shows the **Test Data** cluster.

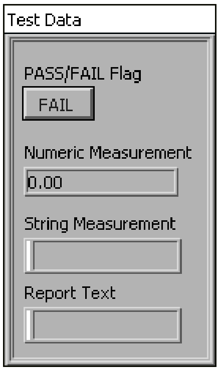


Figure C-1. Test Data Cluster

Table C-1 lists the elements of the **Test Data** cluster, their types, and descriptions of how they are used by the LabVIEW Adapter.

Table C-1. Test Data Cluster Elements

Element	Type	Description
PASS/FAIL Flag	TF	The test VI sets this element to indicate whether the test passed. Valid values are `True` (PASS) or `False` (FAIL). The adapter copies its value into the `Step.Result.PassFail` property if the property exists.
Numeric Measurement	DBL	Numeric measurement that the test VI returns. The adapter copies this value into the `Step.Result.Numeric` property if the property exists.

Table C-1. Test Data Cluster Elements (Continued)

Element	Type	Description
String Measurement	abc	String value that the test VI returns. The adapter copies the string into the `Step.Result.String` property if the property exists.
Report Text	abc	Output message to display in the report. The adapter copies the message value into the `Step.Result.ReportText` property if the property exists.

The LabVIEW Adapter also supports an older version of the **Test Data** cluster from the LabVIEW Test Executive product. The **Test Data** cluster in the LabVIEW Test Executive does not contain a **Report Text** element. Instead, the cluster contains two string elements, **Comment** and **User Output**.

Table C-2 lists the elements of the older **Test Data** cluster, their types, and description of how they are used by the LabVIEW Adapter.

Table C-2. Old Test Data Cluster Elements from the LabVIEW Test Executive

Element	Type	Description
Comment	abc	Output message to display in the report. The adapter copies the message value into the `Step.Result.ReportText` property if the property exists.
User Output	abc	String value that the test VI returns. The adapter dynamically creates the step property `Step.Result.UserOutput`, and copies the string value to the step property.

Error Out Cluster

TestStand must use the contents of the **error out** cluster to determine whether a run-time error has occurred and to take appropriate action, if necessary. When you create a VI, use the standard LabVIEW **error out** cluster, shown in Figure C-2.

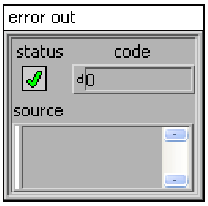


Figure C-2. Standard Error Out Cluster

Table C-3 lists the elements of the **error out** cluster, their types, and descriptions of how they are used by the LabVIEW Adapter.

Table C-3. Error Out Cluster Elements

Element	Type	Description
status	TF	The test VI must set this to `True` if an error occurs. The adapter copies the output value into the `Step.Result.Error.Occurred` property if the property exists.
code	I32	The test VI can set this element to a non-zero value if an error occurs. If the property exists, the adapter copies the output value into `Step.Result.Error.Code`.
source	abc	The test VI can set this element to a descriptive string if an error occurs. If the property exists, the adapter copies the output value into `Step.Result.Error.Msg`.

Input Buffer String Control

Use the **Input Buffer** string control to pass input data directly to the VI. The LabVIEW Adapter automatically copies the contents of the `Step.InBuf` property to the **Input Buffer** control if the property exists.

Invocation Information Cluster

Use the **Invocation Information** cluster to pass additional information to the VI. Figure C-3 shows the **Invocation Information** cluster.

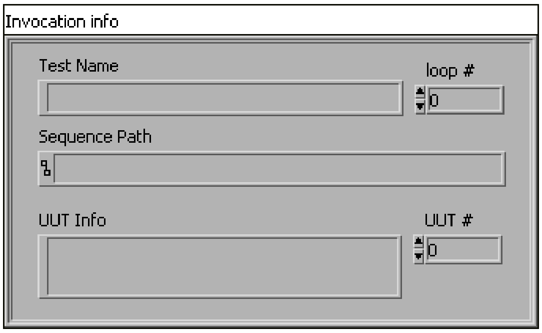


Figure C-3. Invocation Information Cluster

Table C-4 lists the elements of the **Invocation Information** cluster, their types, and descriptions of how the LabVIEW Adapter assigns a value to each cluster element.

Table C-4. Invocation Information Cluster Elements

Element	Type	Description
Test Name	abc	Contains the name of the step that invokes the VI.
loop #	I32	Contains the loop count if the step that invokes the VI is looping on the step.
Sequence Path		Contains the name and absolute path of the sequence file that is running the VI.

Table C-4. Invocation Information Cluster Elements (Continued)

Element	Type	Description
UUT Info	abc	Contains the value of the `RunState.Root.Locals.UUT.SerialNumber` property if the property exists. Refer to Chapter 5, *Configuring the LabVIEW Adapter*, for more information about how to configure this setting.
UUT #	I32	Contains the value of the `RunState.Root.Locals.UUT.UUTLoopIndex` property if the property exists. Refer to Chapter 5, *Configuring the LabVIEW Adapter*, for more information about how to configure this setting.

Sequence Context Control

Use the **Sequence Context** control to obtain a reference to the TestStand SequenceContext object. You can use the sequence context to access all the objects, variables, and properties in the execution. Refer to the *TestStand Help* for more information about using the sequence context from a VI.

D
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 steps in the Cleanup step groups of the sequences in the call stack. When you abort an execution, report generation does not occur.
ActiveX (Microsoft ActiveX)	Set of Microsoft technologies for reusable software components. Formerly called *OLE*.
ActiveX control	A reusable software component that adds functionality to any compatible ActiveX control container.
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	If an adapter is specific to an application development environment (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. Some adapters support stepping into the source code of the ADE while executing the step from the TestStand Sequence Editor.
Application Development Environment (ADE)	A programming environment such as LabVIEW, LabWindows™/CVI™, or Microsoft Visual Basic, 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.
ASCII	American Standard Code for Information Interchange.

B

block diagram	Pictorial description or representation of a program or algorithm. In LabVIEW, the block diagram that consists of executable icons called nodes and wires that carry data between the nodes, is the source code for the VI. The block diagram resides in the Diagram window of the VI.
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 the nested subsequences to complete.
class	Defines a list of methods and properties that you can use with respect to the objects that you create as instances of that class. A class is like a data type definition except that it applies to objects rather than variables.
cluster	A set of ordered, non-indexed data elements in LabVIEW of any data type including numeric, Boolean, string, array, or cluster. The elements must be all controls or all indicators.
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 a particular step type.
connector	Part of a LabVIEW VI or function node that contains its input and output terminals, through which data passes to and from the node.
context menu	Access context menus by right-clicking on an object. Menu options in a context menu pertain specifically to the object you have selected.
control	An input and output device in a panel or window, which you use to enter data or make a setting.

custom property	A property that you define in a step type. Each step you create with the step type has its own copy of the custom property. TestStand uses the value that you specify for the custom property in the step type as the initial value of the property in each new step you create. Normally, after you create a step, you can change the value of the property in the step.

D

dialog box	A user interface, which you use to specify additional information for the completion of a command.
DLL	Dynamic Link Library

E

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.
entry point	A sequence in the process model file that TestStand displays as a menu item, such as Test UUTs, Single Pass, or Report Options.
Execution entry point	A sequence in a process model that runs tests against a UUT. Execution entry points call the Main sequence 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 are visible in the Execute menu. Execution entry points are only visible in the menu when the active window contains a sequence file that contains a Main sequence callback.
expression	A formula that calculates a new value from the values of multiple variables or properties. In expressions, you can access all variables and properties in the sequence context that is active when TestStand evaluates the expression.

F

front panel	The interactive user interface of a LabVIEW VI. Modeled from the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, and other controls and indicators.

G

global variable	TestStand defines two types of global variables: sequence file globals and station globals. Sequence file globals are accessible by any sequence or step in the sequence file. Station globals 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.

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.
LabVIEW Adapter	*See* adapter.
local variable	A property of a sequence that holds a value or additional subproperties. Only a step within the sequence can directly access the property value.
lookup string	A string that defines a complete path from the object on which you call the method to the specific property you want to access.

M

Main sequence	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 main sequences define the steps that are unique to the different types of tests you run.
menu bar	Horizontal bar that contains names of main menus.

N

.NET Adapter	*See* adapter.

O

object	A service that an ActiveX server makes available to clients.
operator interface	A program that provides a graphical user interface (GUI) for executing sequences on a test station.

P

parameters	A value passed to a program or subroutine.
pop-up menus	*See* context menu.
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 be of type number, string, Boolean, container, object reference, a user-defined data type, or an array of these types. 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. Only a container property has the ability to contain any number of subproperties. Each property has a name and a comment.

R

resource string	Text strings stored in an external file so that you can alter the strings without directly altering the application.
run mode	The mode in which you execute a step, such as normal, skip, force pass, or force fail.
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 in the call stack to the top-level sequence.

S

sequence	Located within a sequence file, a sequence contains a series of steps that you specify to execute in a particular order. When and if a step is executed depends 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 (GUI) for creating, editing, and debugging sequences.
sequence file	A file that contains the definition of one or more sequences.
Sequence File window	A separate (child) window within the sequence editor in which a sequence file appears.
soft front panel (SFP)	A graphical display panel for an instrument.
source code template	A set of source files that contain skeleton code, which serves as a starting point for the development of code modules for steps. TestStand uses the source code template when you click **Create Code** on the **Source Code** tab of the Specify Module dialog box for a step.
standard named data type	A data type that TestStand defines and names. You can add subproperties to the standard data types, but you cannot delete any of their built-in subproperties. The standard named data types are `Path`, `Error`, and `CommonResults`.
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.
step	An element that you can insert into a sequence that performs an action, such as calling a code module to perform a specific test. Typically, a sequence contains a series of steps that define your test and execution flow.

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 execute last.
step property	A property of a step.

T

template	*See* code template.
test executive engine	*See* engine.

U

Unit Under Test (UUT)	The device or component that you are testing.

V

variables	A property that you can create in a certain context. 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
VI library	Special file of type `.llb` that contains a collection of related VIs for a specific use.

W

window	A working area that supports specific tasks related to developing and executing programs.
wire	Tool used in LabVIEW to define data paths between source and sink terminals.

Index