NI TestStand™

Using LabVIEW™ with TestStand

April 2007
373200B-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 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 (0) 9 725 72511, France 33 (0) 1 48 14 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

© 2003–2007 National Instruments Corporation. All rights reserved.

Important Information

Warranty

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

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

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

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

Copyright

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

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, ni.com, NI TestStand, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on `ni.com/legal` for more information about National Instruments trademarks.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Conventions

The following conventions are used in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a 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. Italic text 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.

Contents

Chapter 1
Introduction

The Role of LabVIEW in a TestStand-Based System 1-1
- Code Modules 1-1
- Custom User Interfaces 1-2
- Custom Step Types 1-2
- LabVIEW Adapter 1-2

Chapter 2
Calling LabVIEW VIs from TestStand

Required LabVIEW Settings 2-1
LabVIEW Module Tab 2-2
Creating and Configuring a New Step Using the LabVIEW Adapter 2-3

Chapter 3
Creating, Editing, and Debugging LabVIEW VIs from TestStand

Creating a New VI from TestStand 3-1
Editing an Existing VI from TestStand 3-2
Debugging a VI in TestStand 3-3

Chapter 4
Using LabVIEW Data Types with TestStand

Data Type Conversion 4-1
Calling VIs with String Parameters 4-4
Calling VIs with Cluster Parameters 4-4
- Specifying Each Cluster Element Individually 4-5
- Passing Existing TestStand Container Variables to LabVIEW 4-5
- Creating a New Custom Data Type 4-6

Creating TestStand Data Types from LabVIEW Clusters 4-7

Chapter 5
Configuring the LabVIEW Adapter

Selecting a LabVIEW Server 5-1
Per-Step Configuration of the LabVIEW Adapter 5-3
Reserve Loaded VIs for Execution 5-3
Code Template Policy 5-4
Legacy VI Settings 5-6

Chapter 6
Creating Custom User Interfaces in LabVIEW

TestStand User Interface Controls 6-1
TestStand VIs and Functions 6-1
Creating Custom User Interfaces 6-2
Configuring the TestStand UI Controls 6-3
Enabling Sequence Editing 6-3
Handling Events 6-3
Starting TestStand 6-5
Main Event Loop and Shutting Down TestStand 6-5
Menu Bars and Menu Event Handling 6-6
Localization 6-8
Other User Interface Utilities 6-8
Making a Dialog Box Modal to TestStand 6-8
Checking for Stopped Execution 6-9
Running User Interfaces 6-9

Appendix A
Using LabVIEW 8.*x* with TestStand

Using LabVIEW 8.0 .... A-1
LabVIEW 8.0 Real-Time Module Incompatibility A-1
Projects A-1
Project Libraries A-2
Network-Published Shared Variables A-2
Deploying Variables A-2
Using an Aliases File A-3
Project NI-DAQmx Tasks, Channels, and Scales A-3
Conditional Disable Structures and Symbols A-4
64-Bit Integer Data Type A-4
User Access to VI Server A-4
XControls A-4
Remote Execution A-4
Building a TestStand Deployment with LabVIEW 8.0 .... A-4

Appendix B
Calling LabVIEW VIs on Remote Systems

Appendix C
Using the TestStand ActiveX APIs in LabVIEW

Appendix D
Calling Legacy VIs

Appendix E
Technical Support and Professional Services

Index

1

Introduction

This chapter discusses how NI TestStand and NI 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. TestStand 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 user interfaces. You can use LabVIEW to accomplish much of this customization in the following ways:

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

Code Modules

TestStand can call LabVIEW virtual instruments (VIs) with a variety of connector pane configurations. TestStand can call VIs that reside on 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.

Custom User Interfaces

You can use LabVIEW to build custom user interfaces for your test systems and for creating custom sequence editors. Typically, custom user interfaces are 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. If you are using the LabVIEW Full or Professional Development System, you can also create user interfaces using the TestStand UI Controls and the TestStand API. Refer to Chapter 9, *Creating Custom User Interfaces*, of the *NI TestStand Reference Manual* for general information about creating custom user interfaces.

Custom Step Types

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

LabVIEW Adapter

The LabVIEW Adapter offers advanced functionality for calling VIs from TestStand. You can use the LabVIEW Adapter to perform the following tasks in TestStand:

- Call VIs with arbitrary connector panes
- Call VIs on remote computers
- Run VIs in the LabVIEW Run-Time Engine
- Call TestStand VIs from versions of TestStand prior to 3.0 and LabVIEW Test Executive VIs
- Create and edit VIs from TestStand
- Debug VIs (step in/step out) from TestStand
- Run VIs using the LabVIEW Development System
- Run VIs using a LabVIEW executable

2

Calling LabVIEW VIs from TestStand

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

Required LabVIEW Settings

All of the tutorials in this manual require that you have the LabVIEW Development System and TestStand installed on the same computer. In addition, you must configure the LabVIEW Adapter to run VIs using the LabVIEW Development System. Refer to Chapter 5, *Configuring the LabVIEW Adapter*, for more information about configuring these settings for the adapter.

Confirm the following settings in LabVIEW:

- To edit or run a VI from TestStand, you must include the VI in the VI Server: Exported VIs list. By default LabVIEW allows access to all VIs.
- If you use LabVIEW 8.0.1 or earlier, select **Tools»Options»Performance and Disk** to confirm that the **Run with multiple threads** checkbox includes a checkmark to avoid errors when running VIs.

LabVIEW Module Tab

Use the LabVIEW Module tab to configure calls to LabVIEW VIs. To display the LabVIEW Module tab, which is shown in Figure 2-1, select **Specify Module** from the context menu of any step that uses the LabVIEW Adapter.

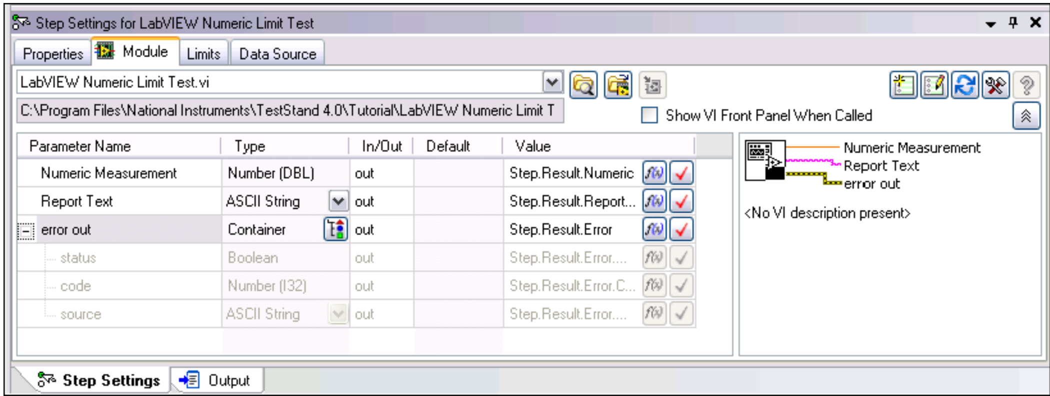


Figure 2-1. LabVIEW Module Tab

The VI Path control on the LabVIEW Module tab contains the name and path of the VI that the adapter executes. The tab also includes buttons for selecting an Express VI, converting an Express VI to a standard VI, creating a new VI based on the code template for the current step type, editing the VI in LabVIEW, refreshing the parameter information for the VI, opening the Advanced Settings window, displaying the help associated with the VI if available, and undocking the LabVIEW Help. You can also specify if LabVIEW displays the front panel of the VI when TestStand calls the VI.

The LabVIEW Module tab also contains specific information about the VI to call, including complete information about the controls and indicators wired to the connector pane of the VI.

- **VI Parameter Table**—Contains the following information about each control and indicator, also called the parameters of the VI, wired to the connector pane of the VI:
  - **Parameter Name**—Contains the caption text for the control or indicator. If no caption exists, this field contains the label text.
  - **Type**—Contains the 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 if the parameter is an input (control) or an output (indicator).
- **Default**—Specifies if TestStand uses the default value for the parameter, cluster element, or array element. If the terminal on the VI is marked Required, this option is not available.
- **Value**—Contains a TestStand expression. For input parameters, TestStand passes the result of this expression to the VI unless you enable the checkbox in the Default column. For output parameters, TestStand stores the data the VI returns in the location this expression specifies.

- **VI Context Help Image and Description**—Displays the context help image of the VI as shown in the LabVIEW Context Help window and displays the description of the VI from the Documentation page of the LabVIEW VI Properties dialog box. When you click a label or terminal of the VI icon, TestStand highlights that parameter in the VI Parameter Table.

Note Click the **Help** or **Help Topic** button located on the Help toolbar to access the *NI TestStand Help*, which provides additional information about the LabVIEW Module tab.

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. Open a new Sequence File window, if one is not already open.
3. Select **File»Save As** and save the sequence file as `<TestStand>\Tutorial\Call LabVIEW VI.seq`.
4. Select **LabVIEW Adapter** in the Insertion Palette.
5. Insert a Pass/Fail step in the Main step group of the Sequence File window. Rename the new step `LabVIEW Pass/Fail Test`.
6. On the LabVIEW Module tab of the Step Settings pane, click the **Browse for VI** button, select `<TestStand>\Tutorial\LabVIEW Pass-Fail Test.vi`, and click **Open**.

7. TestStand reads the description and connector pane information from the VI and updates the LabVIEW Module tab so you can configure the data to pass to and from the VI.
8. In the VI Parameter Table, type `Step.Result.PassFail` in the Value column of the **PASS/FAIL Flag** output parameter and type `Step.Result.ReportText` in the Value column of the **Report Text** output parameter.

Note All expression fields in TestStand support type-ahead and auto-completion through drop-down lists and context-sensitive highlighting. At any point while editing an expression, press **<Ctrl-Space>** to display a drop-down list of valid expression elements.

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

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

Note By default, if a VI uses 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.

10. Select **File»Save** to save the sequence file.
11. 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.

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 `<TestStand>\Tutorial\Call LabVIEW VI.seq`, which you created in the *Creating and Configuring a New Step Using the LabVIEW Adapter* section of Chapter 2, *Calling LabVIEW VIs from TestStand*.
3. Select **LabVIEW Adapter** in the Insertion Palette.
4. Insert a Numeric Limit Test step after the `LabVIEW Pass/Fail Test` step and rename it `LabVIEW Numeric Limit Test`.
5. Select the `LabVIEW Numeric Limit Test` step and click the LabVIEW Module tab of the Step Settings pane.
6. Use the LabVIEW Module tab to complete the following steps:
   a. Click **Create VI** to create a new VI.
   b. In the File dialog box that launches, browse to the `<TestStand>\Tutorial` directory, type `LabVIEW Numeric Limit Test.vi` in the **File Name** control, and click **OK**. TestStand creates a new VI based on the available code templates for the TestStand Numeric Limit Test and opens that VI in LabVIEW.

Note The TestStand Numeric Limit Test step type requires code modules to store a measurement value in the `Step.Result.Numeric` property, and the step type performs a comparison operation to determine if the step passes or fails. Code modules can update step properties by passing step properties as parameters to and from the module or by using the

TestStand API in the module. If you use a default code template from National Instruments to create a module, the parameters needed to access the step properties are created for you.

c. In LabVIEW, select **Window»Show Block Diagram** to open the block diagram.

d. Right-click the **Numeric Measurement** indicator terminal, select **Create»Constant** from the context menu, and type the number `10.0`.

e. Save and close the VI.

7. Return to the TestStand Sequence Editor and select the LabVIEW Module tab. Notice that TestStand automatically updates the output parameters for the VI based on the information stored in the code template for the Numeric Limit Test step type.
8. Save the sequence file as `<TestStand>\Tutorial\Call LabVIEW VI 2.seq`.
9. Start a new execution of 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 `10.0`.

Note For more information about step types and code templates, refer to Chapter 13, *Creating Custom Step Types*, of the *NI TestStand Reference Manual*.

Editing an Existing VI from TestStand

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

1. Open `<TestStand>\Tutorial\Call LabVIEW VI 2.seq`.
2. Right-click the `LabVIEW Pass/Fail Test` step and select **Edit VI** in the LabVIEW Module tab.

   LabVIEW becomes the active application in which `LabVIEW Pass-Fail Test.vi` is open.
3. Open the block diagram for the VI.
4. Change the **PASS/FAIL Flag** Boolean constant to `False`.
5. Save and close the VI.
6. In the TestStand Sequence Editor, start a new execution of the sequence file using the Single Pass Execution entry point.

When the execution is complete, the resulting report shows that the step has failed. The VI now returns `False` in the **PASS/FAIL Flag** 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. Open `<TestStand>\Tutorial\Call LabVIEW VI.seq`.
2. Place a breakpoint on the `LabVIEW Pass/Fail Test` step by clicking to the left of the step. The Stop icon appears to the left of the step where it is set.
3. Select **Execute»Run MainSequence** to start an execution of `MainSequence`.

   The execution starts and then pauses before executing the step.
4. When the execution pauses, click **Step Into** on the sequence editor toolbar.

   LabVIEW becomes the active application, in which the LabVIEW Pass-Fail Test VI is open and in a suspended state.
5. Open the block diagram of the suspended VI.
6. Click **Step Into** or **Step Over** on the LabVIEW toolbar to begin single-stepping through the VI. You can click **Continue** at any time to finish single stepping through the VI.
7. When you have finished single-stepping through the VI, click **Return to Caller** on the LabVIEW toolbar to return to TestStand. The execution then pauses at the next step in the sequence.
8. Select **Debug»Resume** in TestStand to complete the execution.
9. Close the Execution window.

Note You can run the VI multiple times before returning to TestStand. However, LabVIEW passes only the results from the last run 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.

Note TestStand container data types 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. It converts any EXT numbers from LabVIEW into double-precision (DBL) numbers.
64-bit Integer Numeric	TestStand does not support calling VIs with 64-bit Integer Numeric indicators or controls.

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

LabVIEW Data Type	TestStand Data Type
Complex number (CSG, CDB, or CXT)	Number TestStand maps each part of the complex number to separate TestStand Number properties. Refer to the previous information about how TestStand converts EXT numbers.
Enum (U32, U16, or U8)	Number For input parameters, the Value column on the LabVIEW Module tab displays a ring that contains 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 or Automation Refnum	Object reference
.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. If you are using LabVIEW 7.1.1, the objects must be marshallable by ref.
Waveform	LabVIEWAnalogWaveform
Digital Waveform	LabVIEWDigitalWaveform
Digital Data	LabVIEWDigitalData
Picture	String You must select **Binary String** on the LabVIEW Module tab. Refer to the *Calling VIs with String Parameters* section for more information about the String data type.

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

LabVIEW Data Type	TestStand Data Type
Refnum (File I/O, VI, Menu, Queue, TCP connection, and so on)	Number You cannot use references to internal LabVIEW objects inside TestStand or in other types of code modules. You can only store references to LabVIEW objects in TestStand properties and then pass the properties to other VIs.
Timestamp	String Refer to the *Calling VIs with String Parameters* section for more information about the String data type.
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 Refer to the *Calling VIs with String Parameters* section for more information about the String data type.

Note You can use references to external objects, such as ActiveX (Microsoft ActiveX) objects or VISA sessions, 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 LabVIEW strings can contain binary data, including NULL characters, but TestStand strings cannot contain NULL characters.

Use the ring control in the Type column of the VI Parameter Table on the LabVIEW Module tab for String parameters to select ASCII String or Binary String. The default value of the ring control is ASCII String. TestStand does not modify the values of ASCII strings that it passes to or from VIs.

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 escapes the string before storing it and substitutes 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 and substitutes the correct character values for the hexadecimal values in the escaped string.

Calling VIs with Cluster Parameters

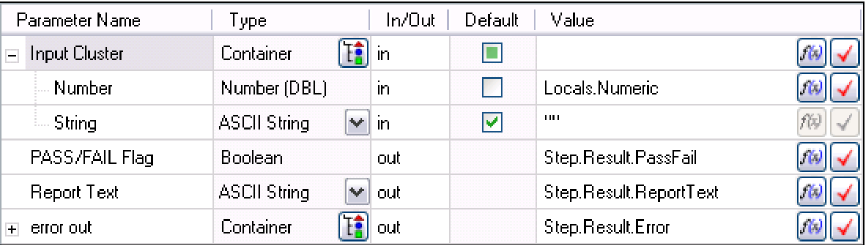
When you configure calls to VIs that use clusters as parameters, you can specify that each cluster element maps to a different TestStand expression, or you can specify that a TestStand data type maps to the entire LabVIEW cluster.

TestStand can also help you create a new custom data type that matches a LabVIEW cluster. For input parameters, you can instruct TestStand to pass the default value for the entire cluster or the default values for specific elements of the cluster control on the front panel of the VI.

For output parameters, you can optionally specify where TestStand should store the cluster value or specific elements of the cluster value.

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 4-1 illustrates a VI Parameter Table in which the data source for the Number element of **Input Cluster** is a local variable. TestStand passes the default value for the String element of Input Cluster.



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

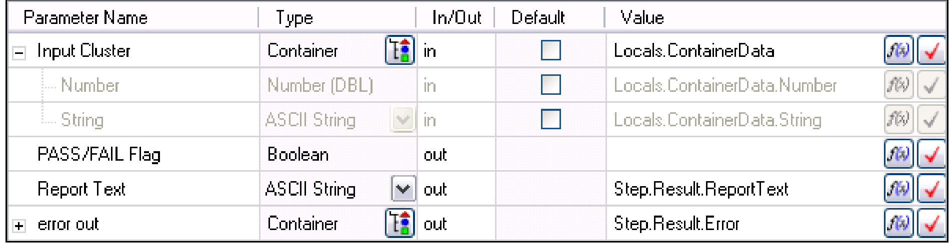
Figure 4-1. Input Cluster Data Sources

Passing Existing TestStand Container Variables to LabVIEW

Instead of passing each cluster element individually, you can create a TestStand custom data type that maps to the entire LabVIEW cluster.

Use the **LabVIEW Cluster Passing** tab of the Type Properties dialog box for the new custom 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 custom data type. Refer to the *NI TestStand Help* for more information about the Type Properties dialog box.

Figure 4-2 illustrates how the data passed to **Input Cluster** is a local variable called ContainerData, with a type of InputData, shown in Figure 4-3.



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

Figure 4-2. ContainerData Local Variable

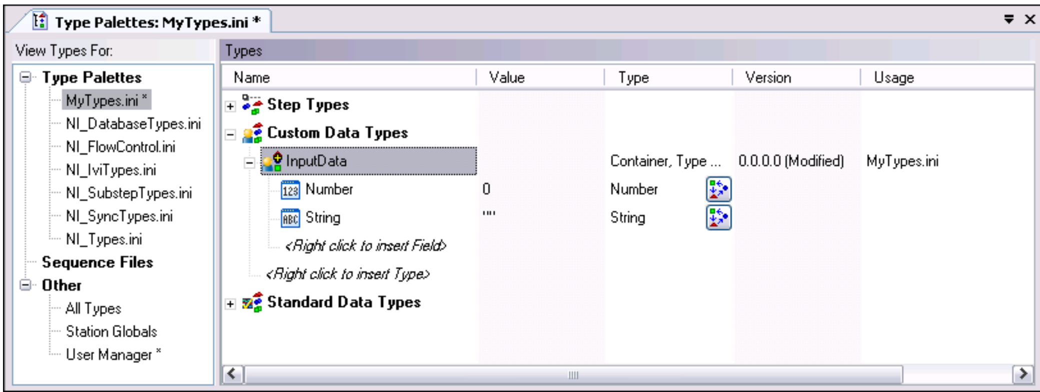


Figure 4-3. TestStand Custom InputData Data Type

Creating a New Custom Data Type

Use the Create Custom Data Type From Cluster dialog box to create a TestStand custom data type, such as a container, that matches an existing LabVIEW cluster. You can click the **Create Custom Data Type** button located in the Type column of the VI Parameter Table on the LabVIEW Module tab to launch the Create Custom Data Type From Cluster dialog box, as shown in Figure 4-4.

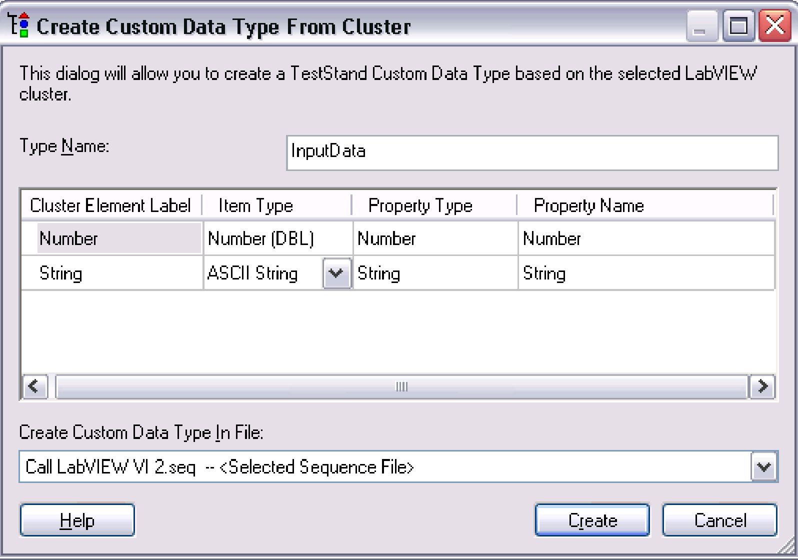


Figure 4-4. Create Custom Data Type From Cluster Dialog Box

Use the Property Type column to specify the name of the TestStand custom data type you want to create. Use the Property Name column to specify the names of the subproperties that map to the elements of the cluster. Use the Create Custom Data Type In File ring control to specify where TestStand creates the type.

Refer to Chapter 11, *Type Concepts*, of the *NI TestStand Reference Manual* for more information about where TestStand stores custom data types. Refer to the *NI TestStand Help* for more information about the Create Custom Data Type From Cluster dialog box.

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.

1. Open `<TestStand>\Tutorial\Call LabVIEW VI 2.seq`.
2. Select **LabVIEW Adapter** in the Insertion Palette.
3. Insert a new Pass/Fail Test step into the Main step group of `MainSequence` after the `LabVIEW Numeric Limit Test` step and rename the step `Pass Container to VI`.
4. On the LabVIEW Module tab, click the **Browse for VI** button and select `<TestStand>\Tutorial\VI with Cluster Input.vi`. The Report Text output parameter of this VI returns a string that contains the string element of the **Input Cluster** parameter.
5. Click the **Create Custom Data Type** button to launch the Create Custom Data Type From Cluster dialog box. TestStand maps the cluster elements to subproperties in a container called Input_Cluster, which is a new TestStand custom 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. Refer to the *NI TestStand Help* and to Chapter 12, *Standard and Custom Data Types*, in the *NI TestStand Reference Manual* for more information about custom data types.
6. In the Create Custom Data Type From Cluster dialog box, change the type name to **InputData** and click **Create** to accept the automatically assigned values and to create the data type in the current sequence file.

7. On the LabVIEW Module tab, remove the checkmark from the Default column for the **Input Cluster** input parameter and click the **Expression Browse** button in the Value column to open the Expression Browser dialog box.
8. Right-click **Locals** and select **Insert Types»InputData** to create a local variable of type `InputData`. Rename the local variable `ContainerData`.
9. Right-click the `Number` subproperty of `Container Data` and select **Properties** from the context menu. Type `23` in the Value field and click **OK**.
10. Right-click the `String` subproperty of `Container Data` and select **Properties** from the context menu. Type `My String Data` in the **Value** field and click **OK**.
11. In the Expression Browser dialog box, enter `Locals.ContainerData` in the **Expression** field and click **OK**. The Value column for the **Input Cluster** parameter now contains `Locals.ContainerData`.
12. Type `Step.Result.ReportText` in the Value column for the **ReportText** output parameter. When TestStand calls the VI, it passes the values in the `ContainerData` local variable to the **Input Cluster** control on the VI and returns the **String** element of the **Input Cluster** parameter to the `ReportText` property of the step.
13. 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, launch the general Adapter Configuration dialog box by selecting **Configure» Adapters**. Select **LabVIEW** in the Adapter column and click **Configure**.

Selecting a LabVIEW Server

The TestStand LabVIEW Adapter can run VIs using 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, shown in Figure 5-1, to select the server you want TestStand to use.

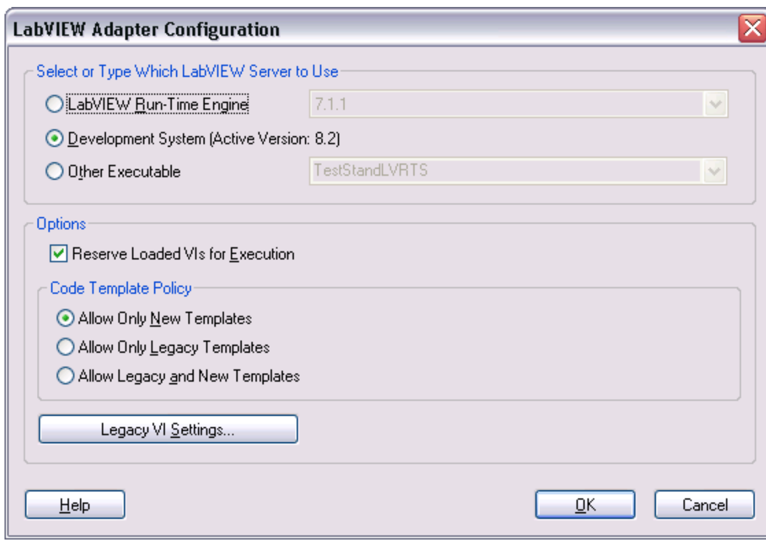


Figure 5-1. LabVIEW Adapter Configuration Dialog Box

- **LabVIEW Run-Time Engine**—Provides optimal performance when calling LabVIEW VIs by running VIs in the same process as TestStand. If you select this option, you cannot create or edit VIs from TestStand or debug VIs that TestStand calls. You must have the LabVIEW Run-Time Engine version that matches the version of the VI on the same computer as TestStand to use this option.
- **Development System (Active Version: *X.X*)**—Allows you to create or edit VIs from TestStand and debug VIs that TestStand calls in LabVIEW. The VIs execute in the LabVIEW Development System process. 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 the Build Executable functionality in LabVIEW 8.0 or later or the Build Application or Shared Library (DLL) functionality in LabVIEW 7.1.1. If you select this option, you cannot create or edit VIs from TestStand or debug VIs that TestStand calls.

  To use this option, enter the ActiveX server name associated with the LabVIEW executable. The executable must be installed and registered on the same computer as TestStand. To register the executable as an ActiveX server, launch the executable once. Refer to `<TestStand>\Components\NI\RuntimeServers\LabVIEW` for an example server VI and application build script for LabVIEW 8.0 or later and for LabVIEW 7.1.1.

Note If you select **LabVIEW Run-Time Engine** or **Other Executable** as the TestStand server, you must fully deploy the VIs before calling them from TestStand. The LabVIEW Run-Time Engine or built executable must be able to find the complete hierarchies of the VIs, including any subVIs. Refer to Chapter 14, *Deploying TestStand Systems*, of the *NI TestStand Reference Manual* for more information about deploying VIs for use with TestStand. Refer to Appendix A, *Using LabVIEW 8.x with TestStand*, for more information about calling and deploying LabVIEW 8.*x* VIs.

Per-Step Configuration of the LabVIEW Adapter

You can direct TestStand to always use the LabVIEW Run-Time Engine to execute a step by selecting the **Always Run VI in LabVIEW Run-Time Engine** option in the Advanced Settings window, which you launch by clicking the **Advanced Settings** button on the LabVIEW Module tab.

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 you last compiled the VI. This setting overrides the global setting in the LabVIEW Adapter Configuration dialog box. Use this option when you create tools and step types for use with the LabVIEW Adapter that you do not want to be affected by the global setting for the adapter.

Reserve Loaded VIs for Execution

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

Although reserving VIs with this option reduces the amount of time required for TestStand to call the VIs, it 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, the Run arrow indicates that the VI is reserved and that you cannot edit the VI. To edit a VI that TestStand has reserved, right-click the step and select **Edit Code** from the context menu

to open the VI in TestStand. You can also select **File»Unload All Modules** in the sequence editor before you open the VI in LabVIEW.

You must close any references you create to VIs. If TestStand reserves VIs when it loads the VIs, LabVIEW does 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 the 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 VI templates are VIs that you can call from previous versions of TestStand. Refer to Appendix D, *Calling Legacy VIs*, for more information about legacy TestStand VIs.

If you have configured the LabVIEW Adapter using the Allow Only New Templates option and then create a new VI from the LabVIEW Module tab, 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. Use the Choose Code Template dialog box, which is illustrated in Figure 5-2, to select the code template to use for the new VI.

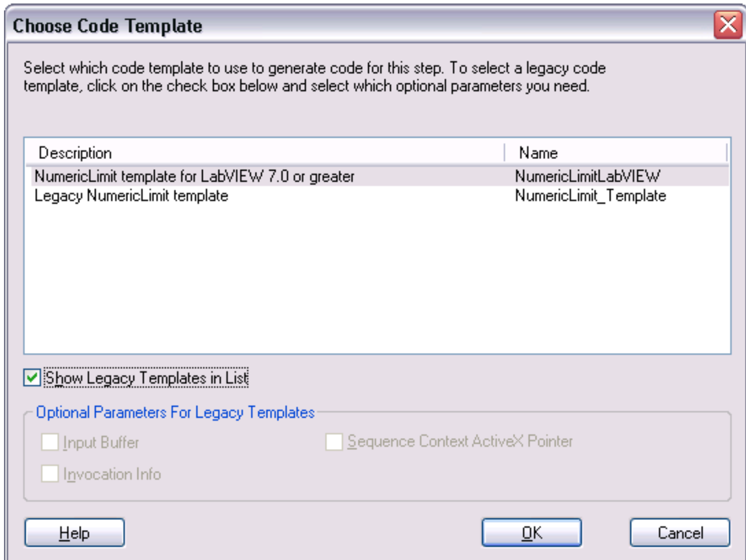


Figure 5-2. Choose Code Template Dialog Box

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, Invocation Info, or Sequence Context ActiveX Pointer, that you want to include as input parameters for the VI.

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

Refer to the *NI TestStand Help* for more information about the Choose Code Template dialog box.

Legacy VI Settings

Click **Legacy VI Settings** on the LabVIEW Adapter Configuration dialog box 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 D, *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 user 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 User Interfaces*, in the *NI TestStand Reference Manual* to obtain a general understanding of the TestStand User Interface (UI) Controls before proceeding with this chapter.

TestStand User Interface Controls

Use the TestStand UI Controls, located on the **Controls»TestStand** palette, to develop a custom user interface application, including custom sequence editors.

When you place the TestStand UI 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. Right-click the control and select **Property Browser** from the context menu to open the LabVIEW Property Browser. Right-click the control and select **Properties** from the context menu to open the property page, if available.

Refer to Appendix C, *Using the TestStand ActiveX APIs in LabVIEW*, for general information about programming the TestStand API from LabVIEW.

TestStand VIs and Functions

The TestStand VIs and functions, located on the **Functions»TestStand** palette, are the LabVIEW versions of the functions in the TestStand Utility Functions Library, or TSUtil.

Use the TestStand VIs and functions for the following tasks:

- Inserting menu items that automatically execute commands the TestStand UI Controls provide.
- Localizing the strings on a user interface.
- Making dialog boxes that LabVIEW VIs launch modal to TestStand applications.
- Checking if an execution that calls a VI has stopped.
- Setting and getting the values of TestStand properties and variables.

Right-click the VI on the **Functions** palette or on the block diagram and select **Help** from the context menu to access the help for the VI.

Creating Custom User Interfaces

User 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

User 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 User Interface using the TestStand UI Controls in LabVIEW, refer to the example user interfaces included with TestStand. Begin with the simple operator interface example, `<TestStand>\UserInterfaces\NI\Simple\LabVIEW\TestExec.llb\Simple OI - Top-Level VI.vi`. For a more advanced sequence editor example that includes menus and localization options, refer to the full-featured example, `<TestStand>\UserInterfaces\NI\Full-Featured\LabVIEW\TestExec.llb\Full UI - Top-Level VI.vi`.

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

Note TestStand no longer includes example user interfaces that use the TestStand API instead of the TestStand UI Controls. These examples contained a large amount of complex source code, and they provided less functionality than the simpler examples that use the TestStand UI Controls. Therefore, National Instruments recommends that you use the new examples that use the TestStand UI Controls as a basis for new development.

Configuring the TestStand UI Controls

Refer to the following example user 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 UI - Configure StatusBar.vi`
- `Full UI - Configure SequenceFileView Manager.vi`
- `Full UI - Configure ListBar.vi`
- `Full UI - Configure ExecutionView Manager.vi`

Enabling Sequence Editing

The TestStand UI Controls support both Operator Mode and Editor Mode. You can instruct the Application Manager control to allow the user to create and edit sequence files by setting the Application.IsEditor property to `True`. You can also set the property by specifying the `/editor` command-line flag.

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.

Complete the following steps to use the Register Event Callback function, available in the LabVIEW Full or Professional Development System:

1. Wire the reference of the control that sends the event you want to handle to the **Event** input of the Register Event Callback function.
2. Use the **Event** input terminal pull-down menu to 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** from the context menu. 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 that contains 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 block diagram of the callback VI to perform the operation you specify when the control generates the event.
7. When the application is finished handling events for the control, it must close the event callback refnum output of the Register Event Callback function by using the Unregister for Events function.

Figure 6-1 illustrates registering a callback VI to handle the Break event for the TestStand Application Manager control.

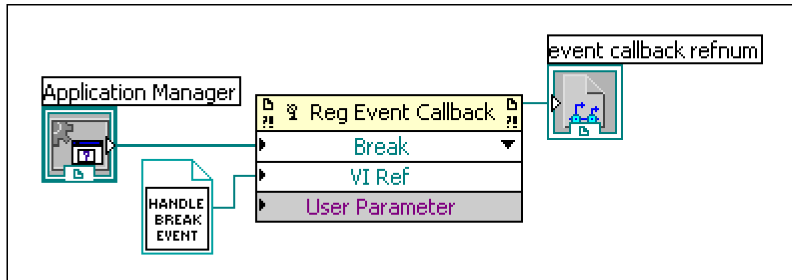


Figure 6-1. Registering a Callback VI for the Break Event

You can 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 user interface VIs for examples of registering and handling events from the TestStand UI Controls:

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

You must limit the tasks that you perform in a callback VI to ensure that LabVIEW handles the event in a timely manner, which allows the front panel to quickly respond to user input and prevents possible hang

conditions. If a callback VI must perform ActiveX operations that may process messages, or perform TestStand operations that could call back into LabVIEW, the application must perform these operations outside of the callback VI. You can defer these operations by defining a user event that the callback VI generates.

Refer to the ReDraw user event in the example full user interface as an illustration of how callback VIs can defer operations that must be performed outside of the callback VI. The example user interface performs the following tasks:

- Calls `Full UI - Create LabVIEW Application Events.vi` to create the ReDraw user event.
- Callback VIs, such as `Full UI - Resized Event Callback.vi`, generate the ReDraw user event when the user interface must resize and reposition controls on the front panel.
- The ReDraw User Event case in the main event loop of the `Full UI - Top-Level VI.vi` sets a global variable while processing the current event to prevent callback VIs from generating new ReDraw events. The ReDraw User Event case calls `Full UI - Disable Panel Updates.vi` to prevent the front panel from updating, calls `Full UI - ArrangeControls.vi` to update the position and size of controls on the front panel, and calls `Full UI - Re-enable Panel Updates.vi` to update the front panel.

Starting TestStand

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

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

Main Event Loop and Shutting Down TestStand

User interface applications wait in a main event loop after starting TestStand. This main event loop must handle the events that stop the user interface application and can handle other events too, such as menu selections and LabVIEW control changes.

Typically, you stop a user interface application by clicking the **Close** box or by executing 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 Panel Close? event. The block diagram that handles the event 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 user interface VIs, informs the main event loop that it can stop.

Refer to the following example user 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 UI - Top-Level VI.vi`
- `Full UI - Create LabVIEW Application Events.vi`
- `Full UI - ExitApplication Event Callback.vi`

Note If you place a TestStand UI Control on the front panel of a VI, you must set the Preferred Execution System to **user interface** for the VI. In addition, National Instruments recommends that if a TestStand User Interface performs ActiveX operations that can process messages, or perform TestStand operations that can call back into LabVIEW, the application must perform these operations in a LabVIEW execution system other than user interface, such as standard or other 2. Performing these operations in the user interface execution system can result in hang conditions.

Menu Bars and Menu Event Handling

The TestStand VI and functions palette contains 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`

Because maintaining the current state of the menu bar can be difficult, you should handle the menu bar only when required. The Event structure in a main event loop can include a case to handle the Menu Activation? event to determine when you open a menu or select a shortcut key that might be linked to a menu item. The block diagram that handles this event can then rebuild the menu bar.

Refer to `Full UI - Top-Level VI.vi` in the example user interface as an illustration of how to rebuild the menu bar as needed. The Menu Activation? case in the main event loop of the VI determines which control has focus and whether the control is a TestStand UI control, and then calls `Full UI - Rebuild Menu Bar.vi` to rebuild the menu bar. When you click on the menu bar, LabVIEW does not automatically return focus to the control after handling a user menu event. The Menu Activation? case in `Full UI - Top-Level VI.vi` passes a reference for the control with focus to the Menu Selection (User) case so that the application can later restore focus to the control.

Add a Menu Selection (User) case to the Event structure in a main event loop to handle user menu selections. You should limit the tasks that you perform in the Menu Selection (User) case to ensure that LabVIEW handles the menu selection in a timely manner. If the case must perform ActiveX operations that may process messages, or perform TestStand operations that could call back into LabVIEW, the application must perform these operations in a LabVIEW execution system other than user interface, such as standard or other 2.

Refer to `Full UI - Top-Level VI.vi` in the example user interface as an illustration of how to process user menu events. The Menu Selection (User) case in the main event loop calls `Full UI - Add To Menu Queue.vi` to queue the operation for processing the menu outside of the main event loop. At the same time as the main event loop, the block diagram calls `Full UI - Process Menu Queue.vi`, which waits for and processes queued operations, and the VI uses the standard LabVIEW execution system. For TestStand menu items, the `Full UI - Process Menu Queue.vi` executes the appropriate TestStand command by calling `TestStand - Execute Menu Command.vi`, and for non-TestStand menu items, the VI calls `Full UI - Process User Menus.vi`, which you can customize to handle your user menu selections.

The LabVIEW application menu items for copy, cut, paste, clear operate on LabVIEW controls only and do not operate on TestStand UI Controls. In addition, the TestStand menu commands operate on TestStand UI Controls only and not LabVIEW controls. When you rebuild a LabVIEW menu in

the Menu Activation? event case and you call the `TestStand - Insert Commands in Menu.vi` function to insert CommandKind_Edit_Copy, CommandKind_Edit_Cut, or CommandKind_Edit_Paste, you can pass `False` to the **TestStand UI Control Has Focus** control and `"Edit"` to the **Top-Level Menu to Insert Into** control of the function to instruct the function to insert the corresponding LabVIEW application menu items instead of the TestStand menu command.

Localization

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

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

Refer to the following example user interface VIs for examples of localizing user interfaces:

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

Other User Interface Utilities

This section outlines some of the functions available in the TestStand palette.

Making a Dialog Box Modal to TestStand

VIs that TestStand calls can launch dialog boxes that are modal to TestStand application windows, such as the TestStand Sequence Editor or custom user interfaces.

Use the following VIs to make a dialog box modal to TestStand application windows: `TestStand - Start Modal Dialog.vi` and `TestStand - End Modal Dialog.vi`.

For a demonstration of how to use these functions, refer to `<TestStand>\Examples\ModalDialogs\LabVIEW`.

Checking for Stopped Execution

VIs that TestStand calls can launch display dialog boxes or perform other time-consuming 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.

Use the following VIs to enable VIs called by TestStand to verify whether the execution that called it has been stopped:

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

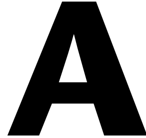
You can also refer to the dialog box VIs in the following examples for a demonstration of how to use these VIs:

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

Running User Interfaces

Consider the following issues when running user interfaces:

- You must close all running LabVIEW User Interfaces before you exit Windows. If you shut down, restart, or log off of Windows while a user interface is running, the user interface cancels the operation and might exit with an error.
- When you run a LabVIEW User Interface in the LabVIEW Development System, you can call remote VIs only if the VI is the same or earlier version as the LabVIEW Development System.
- By default, you can only run one copy of a LabVIEW built executable at a time, which prevents the TestStand `/useexisting` command-line option from working. Add the `"allowmultipleinstances = TRUE"` option to the INI options file in the same directory as your LabVIEW built executable to allow more than one copy to execute at a time.



Using LabVIEW 8.*x* with TestStand

Using LabVIEW 8.0

LabVIEW 8.0 introduces support for many new features, such as projects, project libraries, and DAQmx tasks. Due to some of these changes, LabVIEW 8.0 is not compatible with TestStand 3.1 or earlier. To use LabVIEW 8.0 with TestStand, you must have TestStand 3.5 or later.

This section describes which LabVIEW 8.0 features are supported in TestStand and whether there are any associated limitations. In addition, this section describes the additional requirements necessary to build a TestStand deployment system that includes LabVIEW 8.0 VIs.

LabVIEW 8.0 Real-Time Module Incompatibility

TestStand 4.0 and earlier is not compatible with the LabVIEW 8.0 Real-Time Module. TestStand cannot download VIs to a LabVIEW 8.0 Real-Time Module. Refer to Appendix B, *Calling LabVIEW VIs on Remote Systems*, for more information about executing VIs on LabVIEW 8.0 Real-Time Modules from TestStand.

Projects

When you run a VI in LabVIEW 8.0, the VI runs inside the application instance for a target in a LabVIEW project or the VI runs without a project in the main application instance. In TestStand, the LabVIEW Adapter always runs a VI in the main application instance. Therefore, LabVIEW 8.0 project settings do not apply. Features or settings of a project are not supported in TestStand unless otherwise noted in the following sections.

Project Libraries

LabVIEW project libraries are collections of VIs, type definitions, palette menu files, and other files, including other project libraries. In TestStand, you can call public VIs in a project library. However, you cannot call private VIs. To call a VI in a project library, you must specify the path to the VI file. TestStand does not use qualified paths that include the project library path and name.

Network-Published Shared Variables

LabVIEW 8.0 supports network-published shared variables that allow VIs on distributed systems to share data across the network. LabVIEW identifies shared variables through a network path that includes the computer name (target name), the project library name(s), and the shared variable name. The following sections describe how to use shared variables in VIs that TestStand calls.

Deploying Variables

In LabVIEW, you must deploy a project library that contains the shared variables you want to connect to from within a VI. LabVIEW automatically deploys shared variables from a library when you execute a VI that reads or writes shared variables and the project library containing the shared variables is open in the current LabVIEW project.

Because TestStand executes VIs without a project, LabVIEW cannot automatically deploy shared variables. Therefore, you must do one of the following to deploy shared variables to the local computer:

- Manually deploy the shared variables in the LabVIEW development environment using a project.
- Use the LabVIEW Utility Deploy Library step type, in TestStand, to deploy a project library that contains shared variables. The project library can only contain shared variables; the project library cannot contain any VI files. Refer to the *NI TestStand Help* for more information about the LabVIEW Utility Deploy Library step type.

Note TestStand does not support calling a VI or DLL that programmatically deploys shared variables using the Deploy Library method on a LabVIEW Application reference or using the Deploy Items method on a LabVIEW Project reference. If you programmatically deploy shared variables, the TestStand or LabVIEW application may error and terminate. However, you can use the Deploy Library method from within a standalone executable without adverse effects on TestStand.

Using an Aliases File

When you deploy a project library or execute a VI in TestStand that accesses a shared variable, LabVIEW must determine how to resolve the target name stored in the network path for the variable. When you configure a target in a LabVIEW project, LabVIEW stores the IP address for the target in an aliases file. The aliases file is located in the same directory as the project. The aliases file uses the same base name as the project but uses a `.aliases` file extension.

In TestStand, the aliases file LabVIEW uses is determined by the server you configured the LabVIEW Adapter to use in the LabVIEW Adapter Configuration dialog box.

- **LabVIEW Run-Time Engine**—LabVIEW looks for an aliases file with the same name and location as the TestStand application. For example, `SeqEdit.aliases` for the TestStand Sequence Editor and `TestExec.aliases` for a user interface. You must quit and restart the TestStand application after copying the aliases file from your project directory to the application directory.
- **Development System**—LabVIEW looks for an aliases file, `LabVIEW aliases`, located in the same directory as the LabVIEW development system executable. You must quit and restart LabVIEW after copying the aliases file from your project directory to the LabVIEW directory.
- **Other Executable**—LabVIEW looks for an aliases file with the same name and directory as the LabVIEW executable server. For example, `TestStandLVRTS.aliases` for `TestStandLVRTS.exe` and `TestExec.aliases` for a user interface. You must quit and restart the executable after copying the aliases file from your project directory to the executable directory.

Project NI-DAQmx Tasks, Channels, and Scales

TestStand 4.0 does not support VIs that use NI-DAQmx tasks, channels, and scales that are defined in a project. You must define your NI-DAQmx tasks, channels, and scales in Measurement & Automation Explorer (MAX).

Conditional Disable Structures and Symbols

LabVIEW 8.0 defines a new structure called the Conditional Disable Structure. The Conditional Disable Structure contains one or more subdiagrams, or cases. Depending on the configuration, LabVIEW uses one of the subdiagrams for the duration of the execution. You can specify conditions for the structure based on custom symbols defined in a project. Because TestStand executes VIs in the main application instance, any conditions which use custom symbols will always evaluate to `False`.

64-Bit Integer Data Type

TestStand does not support calling VIs that contain terminals connected to 64-bit Integer Numeric indicators or controls.

User Access to VI Server

TestStand does not support user-based VI security for executing VIs on remote systems. You must use machine access security to protect a VI server on a remote system.

XControls

TestStand supports calling VIs that use XControls on Windows systems.

Remote Execution

To execute VIs on a remote LabVIEW system, TestStand requires Remote Execution Support for NI TestStand for the latest version of LabVIEW on your system. The version of this component must match the version of LabVIEW with which you saved your VIs. LabVIEW installs this component only if TestStand is present. If you install TestStand after you install LabVIEW, TestStand installs a version of the component that may not match the version of LabVIEW on your system. In this case, you may need to rerun the LabVIEW installer.

Building a TestStand Deployment with LabVIEW 8.0

TestStand 4.0 supports deploying LabVIEW 8.0 VIs and VIs from project libraries. However, the following new restrictions exist that do not apply to earlier versions of LabVIEW:

- TestStand does not support deployment of duplicate project libraries. You should not attempt to include two project libraries with the same name.

- TestStand no longer supports including two VIs with the same name, unless the VIs are included in different project libraries.
- National Instruments does not recommend distributing two VIs with the same name to different locations on a target machine. If LabVIEW attempts to load a subVI, when a VI with the same name but from a different location is already in memory, LabVIEW uses the VI in memory. However, LabVIEW reports an error if you attempt to load a top-level VI, even if the similarly named VI in memory is an exact copy of the one you want to load.
- TestStand no longer supports distributing duplicate DLLs that are called by VIs. TestStand 4.0 still supports, but does not recommend, distributing duplicate DLLs that are called by steps in a sequence file.
- National Instruments does not recommend editing packaged VIs on a deployed system because unexpected errors can occur. For example, TestStand only includes required VIs from a project library in a deployment. If you attempt to add new VIs from a project library to the deployment image, the new VIs may not be found by LabVIEW when executed. Analysis and instrument drivers are examples of VIs that use project libraries.

  National Instruments recommends that you rebuild and redeploy the deployment image on the LabVIEW Development System.
- National Instruments does not recommend deploying LabVIEW VIs that were previously deployed using the TestStand Deployment Utility. When the TestStand Deployment Utility packages LabVIEW VIs, the utility includes all subVIs, and creates partial project libraries that contain only required VIs from the project libraries that the VIs need.

  If you attempt to redeploy these VIs, the build in the TestStand Deployment Utility can fail when attempting to include a partial project library and the original complete project library, or when attempting to include two copies of the same VI on the system. For example, if you deploy custom LabVIEW based step types to a development system and then attempt to include the step types in a new deployment built by that system, the build could fail.

  To work around this limitation, you must remove the duplicate VIs and partial project libraries from the system and relink the VIs to the original VIs and complete project libraries on the system before building a new deployment. Use the following strategy to resolve any duplicate VIs from previously deployed files:

  1. Build the deployment and review which VIs the utility reports as duplicates in the status log.

2. Review the SupportVIs LLB or directory in the previously deployed files to determine which VIs and project libraries will conflict with VIs located on your development system. Typically, a SupportVIs LLB or directory contains duplicate VIs and project libraries from `vi.lib` and `user.lib`.
3. Remove any duplicate VIs and partial project libraries reported by the Deployment Utility and those in a SupportVIs LLB or directory.
4. Load all top-level VIs included in the previously deployed files and resave the VIs. LabVIEW will prompt you to browse for any subVIs that are not found.
5. If the previously deployed files contain sequence files with steps that call Express VIs or any duplicate VI, you must reconfigure the Express VI steps and update the VI pathname for steps that call VIs.
6. Repeat step 1 to determine if duplicate VIs still exist.

While processing sequence files, the TestStand Deployment Utility automatically includes project library files in the deployment if the project library is referenced by a LabVIEW Utility Deploy Library step.

When you build a TestStand deployment that calls VIs that use shared variables, you must add an aliases file to the deployment. You must also configure the deployment to install the aliases file to the proper location on the destination system so that LabVIEW can properly resolve network paths. In addition, you must ensure that the shared variables can be deployed by the destination system by including the NI Variable Engine component in the installer.

LabVIEW Object-Oriented Programming

LabVIEW 8.2 added support for Object-Oriented Programming. TestStand cannot directly call a VI that wires a LabVIEW object to its connector pane. However, you can wire the LabVIEW object to a Flatten to String primitive, return the data string to TestStand as a binary string, and store the flattened LabVIEW object in a string property or variable in TestStand.

You cannot access the properties or invoke the methods on the flattened LabVIEW string object in TestStand. To access the properties or invoke the methods on the LabVIEW object, you must pass the flattened LabVIEW object as a binary string to a VI and wire the string to the Unflatten From String primitive along with the object constant of the correct type to create the LabVIEW object.

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. TestStand 4.0 supports downloading VIs to systems running the LabVIEW 7.1.1 RT module but does not support downloading VIs to systems running the LabVIEW 8.0 or later RT module. Instead, you must call a VI on the local system and the local VI must then call the VI on the LabVIEW RT module, or you must manually download the VI to the RT Module and call the VI using its remote path or by name, if the VI is already in memory.

Because TestStand uses the LabVIEW VI Server to run VIs remotely, these remote computers can use any operating system that LabVIEW supports, including Linux, Solaris (LabVIEW 7.1.1 only), 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 located on that computer. You must also configure the computer running TestStand to have network access to the remote computer running the LabVIEW VI Server.

Configuring a Step to Run Remotely

Complete the following steps to configure a step to run remotely:

1. Click **Advanced Settings** on the LabVIEW Module tab to launch the Advanced Settings window, which you use to specify the name, or an expression that evaluates to the name, of the remote computer on which you want to run the VI.
2. If the remote computer is running the LabVIEW Development System or a LabVIEW executable, use the Remote VI Path text box to specify the path to the VI on the remote computer.

   If the remote computer is a PXI controller running LabVIEW 7.1.1 RT, TestStand downloads the VI to the remote computer or loads the VI

using the Remote VI Path that you specified in the Advanced Settings window. 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.

If the remote computer is a PXI controller running LabVIEW 8.0 RT or later, TestStand does not download the VI to the remote computer.

You can use the LabVIEW 8.0 Development System to download VIs to the PXI controller.

Note For more information about downloading VIs to a PXI controller, refer to the *Getting Started with the LabVIEW Real-Time Module* manual in the `<LabVIEW>\manuals` folder.

You can also use FTP to download VIs to the PXI controller. Using the LabVIEW Development System, you can create a Source Distribution with all your VIs. This ensures that all of the dependencies of the VI will be included so that it can be transferred to the hard drive of the RT target. Be sure to clear the check marks from the options to exclude VIs from `vi.lib`, `instr.lib`, and `user.lib`. Then, you can use FTP protocol, to transfer the source distribution output to the hard drive of the RT target.

Once the VIs are downloaded to the PXI controller running LabVIEW 8.0 RT module or greater, you can call a VI by specifying the Remote VI Path in the Advanced Settings window.

Note The VI must be present on the local computer so TestStand can configure and run the VI.

Configuring the LabVIEW VI Server to Run VIs Remotely

The LabVIEW Development System or built executable must be running on the remote machine and configured to allow VI calls through the TCP/IP protocol of the VI Server.

In LabVIEW 7.1.1, select **Tools»Options** and navigate to the VI Server: Configuration settings to enable the TCP/IP protocol and the VI calls options. You can also specify the TCP/IP port that the server uses. The port you specify in LabVIEW must be the same port that you specify in TestStand in the Advanced Settings window, accessible from the LabVIEW Module tab.

Use the VI Server: TCP/IP or Machine Access settings to allow specific computers access to the LabVIEW VI Server. You can also specify specific computers or entire domains that can call VIs on the server machine.

Use the VI Server: Exported VIs settings to configure the VIs that you want to call through the LabVIEW VI Server.

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

In LabVIEW 8.0 or later, you can make changes to the LabVIEW VI Server settings using LabVIEW Projects. You must enable TCP/IP Protocol and specify the Machine Access, User Access, Exported VIs settings.

Refer to the *LabVIEW Help* for more information about configuring VI Server options.

Configuring the LabVIEW RT Server to Run VIs

You can configure an RT Server to Run VIs remotely similarly to the configuration described in the Configuring the LabVIEW VI Server to Run VIs Remotely section.

For LabVIEW 7.1.1 RT Server, launch LabVIEW on the host computer, select the appropriate RT target computer, and select **Tools»RT Target <IP Address/Host Name> Options**. Configure the VI Server settings to specify which computers and VIs can run remotely.

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.

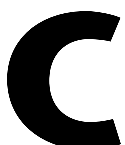
For LabVIEW 8.0 and later RT Server, launch LabVIEW on the host computer, select the appropriate RT target in the project, and select **Properties** from the context menu to launch the Real-Time PXI Properties dialog box. Configure the VI Server settings to specify which computers and VIs can run remotely.

Note When you finish configuring the target computer, untarget the PXI controller before attempting to use TestStand to call VIs on the target computer.

Note For more information about configuring VI Server settings on a PXI controller, refer to the *Configuring Target Properties* section of the *Getting Started with the LabVIEW Real-Time Module* manual in the `<LabVIEW>\manuals` folder.

User Access to VI Server

TestStand does not support user-based VI security for executing VIs on remote systems. You must use machine access security to protect a VI server on a remote system.



Using the TestStand ActiveX APIs in LabVIEW

In some cases you may 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.

Refer to the LabVIEW documentation for fundamental information about ActiveX concepts and how to use LabVIEW as an ActiveX client. National Instruments recommends that you become familiar with that 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. The block diagram in Figure C-1 illustrates invoking the UnloadModules method of a Sequence object to unload all code modules in the sequence.

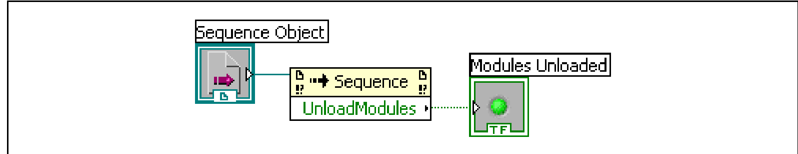


Figure C-1. Invoking the UnloadModules Method

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.

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

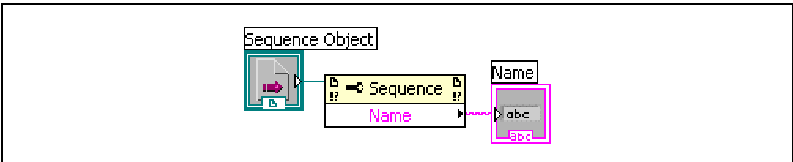


Figure C-2. Obtaining the Value of the Name Property from a Sequence Object

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

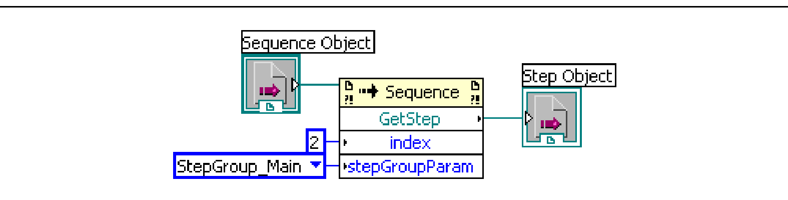


Figure C-3. Obtaining a Reference to a Step of a Sequence Referenced by a Sequence Object

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 AsPropertyObject method of the object. 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 C-4 illustrates using the GetValString method on the PropertyObject interface of a Step object to obtain the error message value for the current step.

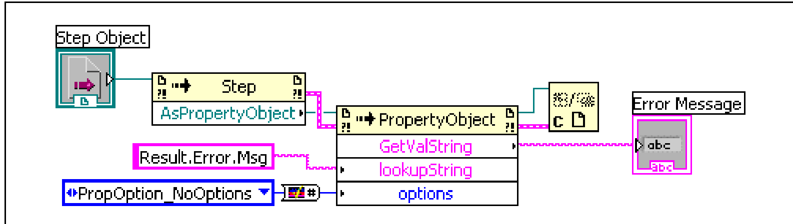


Figure C-4. Using the GetValString Method to Obtain the Error Message Value for the Current Step

You can access dynamic properties of a SequenceContext object by using `TestStand - Get Property Value.vi` or `TestStand - Set Property Value.vi`.

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

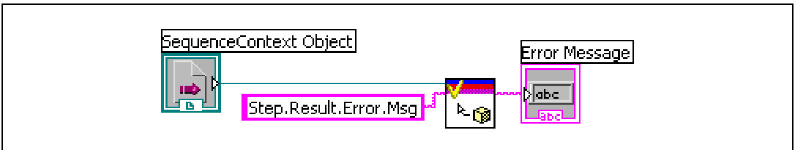


Figure C-5. Obtaining the Error Message for the Current Step

Releasing ActiveX References

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

Note If you do not release the ActiveX reference, LabVIEW does not release it for you until the 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 acceptable values of these arguments are organized into groups that correspond to different properties and methods. For example, the SetValNumber method on the PropertyObject class has an options input argument that accepts many different numeric constants.

It can be difficult to remember all 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`.

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 you can use with TestStand API properties and methods. Use both of these VIs in conjunction with the information in the *NI TestStand Help* regarding constants associated with the TestStand API methods and properties.

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 C-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 enumeration input arguments. For these methods, right-click the input parameter on the Invoke Node in LabVIEW, select **Create»Constant** to create a LabVIEW ring constant, and select the value you want in the resulting constant.

Getting a Different Interface for a TestStand Object

In some cases, you might 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 the LabVIEWModule interface instead, perform a

QueryInterface on the Module object to obtain that interface. In LabVIEW, use the Variant To Data function with the reference to accomplish this task.

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

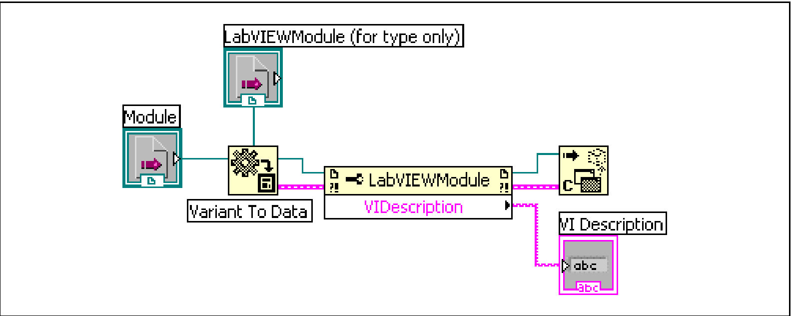


Figure C-6. Converting a Module Reference to the LabVIEWModule Type

Acquiring a Derived Class from the PropertyObject Class

In some cases, you might need to obtain a reference to a TestStand object using the PropertyObject class methods. You might 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 C-7 illustrates obtaining a reference to a Step object from a SequenceContext object using a lookup string.

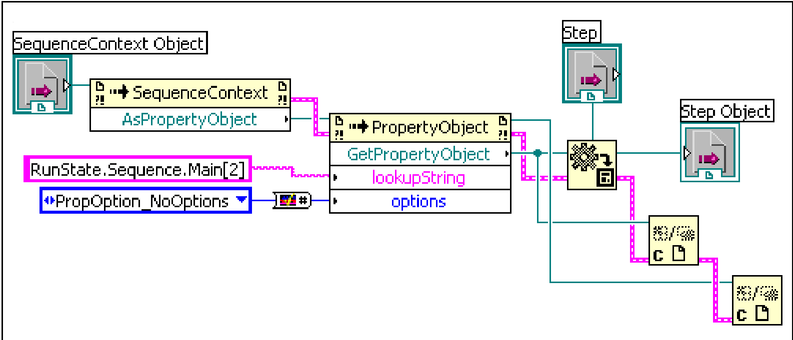


Figure C-7. Obtaining a Reference to a Step Object from a SequenceContext Object Using a Lookup String

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.

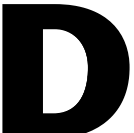
For example, you can have a LabVIEW code module that calls the Engine.NewExecution method followed by the Engine.WaitForEnd method, and a new execution that calls LabVIEW code modules. Deadlock can occur if either of the VIs in this scenario use Same As Caller or User Interface as its preferred execution 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.

Note TestStand no longer includes example user interfaces that use the TestStand API instead of the TestStand UI Controls. If you attempt to create a user interface that uses the TestStand API instead of the TestStand UI Controls, National Instruments recommends that you set the LabVIEW Preferred Execution System for its VIs to **Other 2**, instead of **User Interface** or **Same as Caller**, to ensure that TestStand ActiveX API calls do not result in a deadlock.

Handling Events

TestStand controls can generate events to notify your application of user input and application events, such as the completion of an execution. To handle events in LabVIEW, register a callback VI using the Register Event Callback function and then use the Unregister for Events function to close the callback before closing your application.

Refer to Chapter 6, *Creating Custom User Interfaces in LabVIEW*, for more information about handling events that TestStand UI Controls generate.



Calling Legacy VIs

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

Format of Legacy VIs

All legacy-style VIs must include the **Test Data** cluster and **error out** cluster indicators. The **Input Buffer**, **Invocation Info**, and **Sequence Context** controls are optional inputs to legacy VIs, which can contain any combination of these controls.

You must assign each control and indicator of the test VI to a terminal on the connector pane of the test VI. If these assignments do not exist, TestStand returns an error when it attempts to call the test VI. TestStand does not require that you use a particular connector pane pattern, and it does not require that you assign the controls and indicators to specific terminals.

While you would usually create new VIs using the LabVIEW Module tab 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.

You can use the following two methods to pass data between your code module and TestStand.

- Using the **Test Data** cluster.
- Using the sequence context ActiveX reference. This method allows you to call the TestStand ActiveX API functions to set the variables used to store the results of your test, such as Step.Result.PassFail.

Note The values set using the sequence context ActiveX reference take precedence over the values set using the **Test Data** cluster. In other words, if you use both methods to set the value of the same variable, the values you set using the sequence context ActiveX reference are recognized. The values you set using the **Test Data** cluster are ignored. You can use both the sequence context ActiveX reference and the **Test Data** cluster together in your code module provided that you do not try to set the same variable twice. For example, if you use the sequence context ActiveX reference to set the value of Step.Result.PassFail

and then use the **Test Data** cluster to set the value of Step.Result.ReportText, both values are set correctly.

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 uses the **Test Data** cluster to return result data from the VI to TestStand, which then uses the data to make a PASS/FAIL determination.

Table D-1 lists the elements of the **Test Data** cluster, the data type of the cluster element, and descriptions of how the LabVIEW Adapter uses each cluster element.

Table D-1. Test Data Cluster Elements

Cluster Element	Data Type	Description
PASS/FAIL Flag	TF	The test VI sets this element to indicate if the test passed. Valid values are `True` (PASS) or `False` (FAIL). The adapter copies the 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.
String Measurement	abc	String value that the test VI returns. The adapter copies this string into the Step.Result.String property if the property exists.
Report Text	abc	Output message to display in the report. The adapter copies this 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 LabVIEW Test Executive **Test Data** cluster does not contain a **Report Text** element but instead contains two string elements, **Comment** and **User Output**.

Table D-2 lists the elements of the older **Test Data** cluster, the data type of the cluster element, and descriptions of how the LabVIEW Adapter uses each cluster element.

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

Cluster Element	Data Type	Description
Comment	abc	Output message to display in the report. The adapter copies this 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 uses the contents of the **error out** cluster to determine if a run-time error occurs and to take appropriate action, if necessary. When you create a VI, use the standard LabVIEW **error out** cluster.

Table D-3 lists the elements of the **error out** cluster, the data type of the cluster element, and descriptions of how the LabVIEW Adapter uses each cluster element.

Table D-3. Error Out Cluster Elements

Cluster Element	Data 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. The adapter copies the output value into the Step.Result.Error.Code property if the property exists.
source	abc	The test VI can set this element to a descriptive string if an error occurs. The adapter copies the output value into the Step.Result.Error.Msg property if the property exists.

Input Buffer String Control

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

Invocation Info Cluster

Use the **Invocation Info** cluster to pass additional information to the VI.

Table D-4 lists the elements of the **Invocation Info** cluster, the data type of the cluster element, and descriptions of how the LabVIEW Adapter uses each cluster element.

Table D-4. Invocation Info Cluster Elements

Cluster Element	Data Type	Description
Test Name	abc	The adapter uses the name of the step that invokes the test VI.
loop #	I32	The adapter uses the loop count if the step that invokes the test VI loops on the step.
Sequence Path		The adapter uses the name and absolute path of the sequence file that runs the test VI.
UUT Info	abc	The adapter uses the value from the RunState.Root.Locals. UUT.SerialNumber property if the property exists. Otherwise, the adapter copies an empty string. Refer to Chapter 5, *Configuring the LabVIEW Adapter*, for more information about how to configure this setting.
UUT #	I32	The adapter uses the value from the RunState.Root.Locals. UUT.UUTLoopIndex property if the property exists. Otherwise, the adapter copies an empty string. 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 *NI TestStand Help* for more information about using the sequence context from a VI.

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 at ni.com/support include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Discussion Forums at ni.com/forums. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit ni.com/services or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. 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, National Instruments Alliance Partner 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.

Index

A

accessing properties
- built-in, C-1
- dynamic, C-2

acquiring a derived class from the PropertyObject class, C-5
ActiveX references, releasing, C-3
aliases file, using, A-3

C

calling VIs
- LabVIEW VIs, 2-1
- LabVIEW VIs, on remote systems, B-1
- legacy VIs, D-1
- with cluster parameters, 4-4
- with string parameters, 4-4

Choose Code Template dialog box, 5-5
clusters
- Cluster Passing tab, 4-5
- Error Out, D-4
- Invocation Info, D-5
- LabVIEW, 4-1, 4-5
- specifying cluster elements individually, 4-5
- Test Data, D-2

code modules, 1-1
code template policy
- Allow Legacy and New Templates option, 5-5
- Allow Only Legacy Templates option, 5-5
- Allow Only New Templates option, 5-4

configuring
- LabVIEW Adapter, 5-1
- LabVIEW RT Server, B-3
- LabVIEW VI Server, B-2
- new step with LabVIEW Adapter, 2-3
- per-step configuration of LabVIEW Adapter, 5-3
- remote steps, B-1
- TestStand UI Controls, 6-3

controls
- Input Buffer String control, D-4
- Sequence Context control, D-6
- TestStand UI controls, 6-3

conventions used in the manual, *iv*
converting data types, 4-1
Create Custom Data Type From Cluster dialog box, 4-6
creating
- custom data type, 4-6
- custom user interfaces, 6-1, 6-2
- new step with the LabVIEW Adapter, 2-3
- new VIs, 3-1
- TestStand data types, 4-6
  - from LabVIEW clusters, 4-7

custom
- step types, 1-2
- user interfaces, 1-2
- user interfaces, creating, 6-1, 6-2

D

data types
- 64-bit integer, A-4
- converting data types, 4-1
- creating a custom data type, 4-6
- creating TestStand data types from LabVIEW clusters, 4-7
- TestStand built-in data types, 4-1
- using LabVIEW data types with TestStand, 4-1

debugging VIs, 3-3

deploying variables, A-2
diagnostic tools (NI resources), E-1
dialog box
    Choose Code Template, 5-5
    Create Custom Data Type from Cluster, 4-6
    LabVIEW Adapter Configuration, 5-1
        Code Template Policy, 5-4
    modal to TestStand, 6-8
documentation
    conventions used in the manual, *iv*
    NI resources, E-1
drivers (NI resources), E-1
drivers, NI resources, E-1

E

editing
    sequences, 6-3
    VIs, 3-2
Error Out cluster, D-4
event handling, 6-3, C-7
examples, NI resources, E-1
execution checking, 6-9
execution system, setting preferred, C-6

H

handling
    events, 6-3, C-7
    menu events, 6-6
help, technical support, E-1

I

Input Buffer string control, D-4
instrument drivers (NI resources), E-1
interfaces for TestStand objects, C-4
introduction, 1-1
Invocation Info cluster, D-5
invoking methods, C-1

K

KnowledgeBase, E-1

L

LabVIEW
    Adapter Configuration dialog box, 5-1
    calling VIs from TestStand, 2-1
    calling VIs, on remote systems, B-1
    cluster, 4-5
    configuring the LabVIEW Adapter, 5-1
    creating custom user interfaces, 6-1
    creating new VIs in TestStand, 3-1
    creating, editing, and debugging VIs in TestStand, 3-1
    debugging VIs in TestStand, 3-3
    deploying variables, A-2
    editing an existing VI in TestStand, 3-2
    LabVIEW Adapter, 2-3, 3-1, 3-3, 5-1
    object oriented programming, A-6
    preferred execution system, C-6
    project libraries, A-2
    remote execution, A-4
    required settings, 2-1
    RT Server, configuring, B-3
    server, selecting, 5-1
    using LabVIEW 8.0, A-1
    using LabVIEW with TestStand, 1-1
    using the TestStand ActiveX APIs, C-1
    VI Server, configuring, B-2
    VIs
        calling, 2-1
        debugging, 3-3

LabVIEW 8.0, A-1
- aliases file, A-3
- building a TestStand deployment with, A-4
- conditional disable structure, A-4
- network-published shared variables, A-2
- projects, A-1
- real-time module incompatibility, A-1
- XControls, A-4

LabVIEW 8.*x* and TestStand, using, A-1
LabVIEW Adapter, 1-2, 2-3, 3-1, 3-3
- configuring, 5-1
- creating and configuring a new step using the LabVIEW Adapter, 2-3
- per-step configuration, 5-3
- setting preferred execution system, C-6

LabVIEW Adapter Configuration dialog box
- Code Template Policy, 5-4

LabVIEW Module tab, 2-2, 4-6
- VI Parameter Table, 2-2, 4-4

legacy VIs
- calling legacy VIs, D-1
- format of legacy VIs, D-1
- settings, 5-6

localization, 6-8

M

making dialog boxes modal to TestStand, 6-8
menu bars, 6-6
menu event handling, 6-6

N

NI support and services, E-1
NI-DAQmx tasks, channels, scales, A-3

O

object-oriented programming, A-6

P

passing TestStand container variables to LabVIEW, 4-5
programming examples (NI resources), E-1
project libraries, A-2
properties, accessing
- built-in, C-1
- dynamic, C-2

R

Register Event Callback function, 6-3
releasing ActiveX references, C-3
remote execution, A-4
remote systems
- calling LabVIEW VIs, B-1
- configuring a LabVIEW VI Server, B-2
- configuring a LabVIEW RT Server, B-3
- configuring a step, B-1

required LabVIEW settings, 2-1
reserving loaded VIs for execution, 5-3
running user interfaces, 6-9

S

selecting a LabVIEW server, 5-1
Sequence Context control, D-6
sequence editing, 6-3
setting the preferred execution system for LabVIEW VIs, C-6
shutting down TestStand, 6-5
software (NI resources), E-1
starting TestStand, 6-5
step types, custom, 1-2
stopped executions, 6-9
string parameters, calling VIs, 4-4

T

technical support, E-1
Test Data cluster, D-2
TestStand
  ActiveX APIs, C-1
  constants and enumerations, C-4
  creating, editing, and debugging VIs, 3-1
  starting, 6-5
  using LabVIEW data types, 4-1
  using with LabVIEW 8.*x*, A-1
  VIs and functions, 6-1
TestStand UI Controls
  configuration, 6-3
  creating custom user interfaces, 6-2
  editing sequences, 6-3
  handling events, 6-3
  introduction, 6-1
  localization, 6-8
  main event loop, 6-5
  menu bars, 6-6
  shutting down TestStand, 6-5
  starting TestStand, 6-5
TestStand Utility Functions Library, 6-1
training and certification, NI resources, E-1
troubleshooting, NI resources, E-1
Tutorials
  Configuring a LabVIEW RT Server to Run VIs, B-3
  Configuring a LabVIEW VI Server to Run VIs Remotely, B-2
  Configuring a Step to Run Remotely, B-1
  Creating a New VI from TestStand, 3-1
  Creating and Configuring a New Step Using the LabVIEW Adapter, 2-3
  Creating TestStand Data Types from LabVIEW Clusters, 4-7
  Debugging a VI in TestStand, 3-3
  Editing an Existing VI from TestStand, 3-2

U

user interface utilities, 6-8
user interfaces
  custom, 1-2, 6-1, 6-2
  running, 6-9
using LabVIEW data types with TestStand, 4-1
using LabVIEW in a TestStand system, 1-1
using TestStand API constants and enumerations, C-4
using the TestStand ActiveX APIs, C-1

V

variables, deploying, A-2
VI Parameter Table, 2-2, 4-4
VI Server
  configuring, B-2
  user access, A-4, B-4
VIs
  calling, 2-1
  debugging, 3-3

W

Web resources, E-1