NI TestStand™

Getting Started with TestStand

August 2014
373436H-01

NATIONAL INSTRUMENTS™

Important Information

Patents

Export Compliance Information

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

Contents

1

Introduction to TestStand

NI TestStand is a flexible and open test management framework for building, customizing, and deploying a full-featured test management system.

If you are using TestStand for the first time, National Instruments recommends that you complete the tutorials in this manual. These tutorials begin with a general introduction to the TestStand Sequence Editor and continue by teaching you how to build sequences, manage test data, and customize TestStand functionality. Because the steps of the tutorials depend on elements from previous tutorials, National Instruments recommends that you follow the chapter outline in order. Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Note The `<TestStand Public>` directory is located by default at `C:\Users\Public\Documents\National Instruments\TestStand` on Microsoft Windows 8.1/8/7/Vista and at `C:\Documents and Settings\All Users\Documents\National Instruments\TestStand` on Windows XP.

TestStand Documentation and Resources

Refer to the *NI TestStand Help* for more information about TestStand features covered in this manual. The TestStand documentation assumes you are familiar with the concepts and tutorials in this manual. You can access the *NI TestStand Help* in the following ways:

- Select **Help»NI TestStand Help** in the sequence editor.
- **(Windows 8.1/8)** Click the **NI Launcher** tile on the Start screen and select **TestStand»TestStand Documentation»TestStand Help**.
- **(Windows 7 or earlier)** Select **Start»All Programs»National Instruments»TestStand»TestStand Documentation»TestStand Help**.

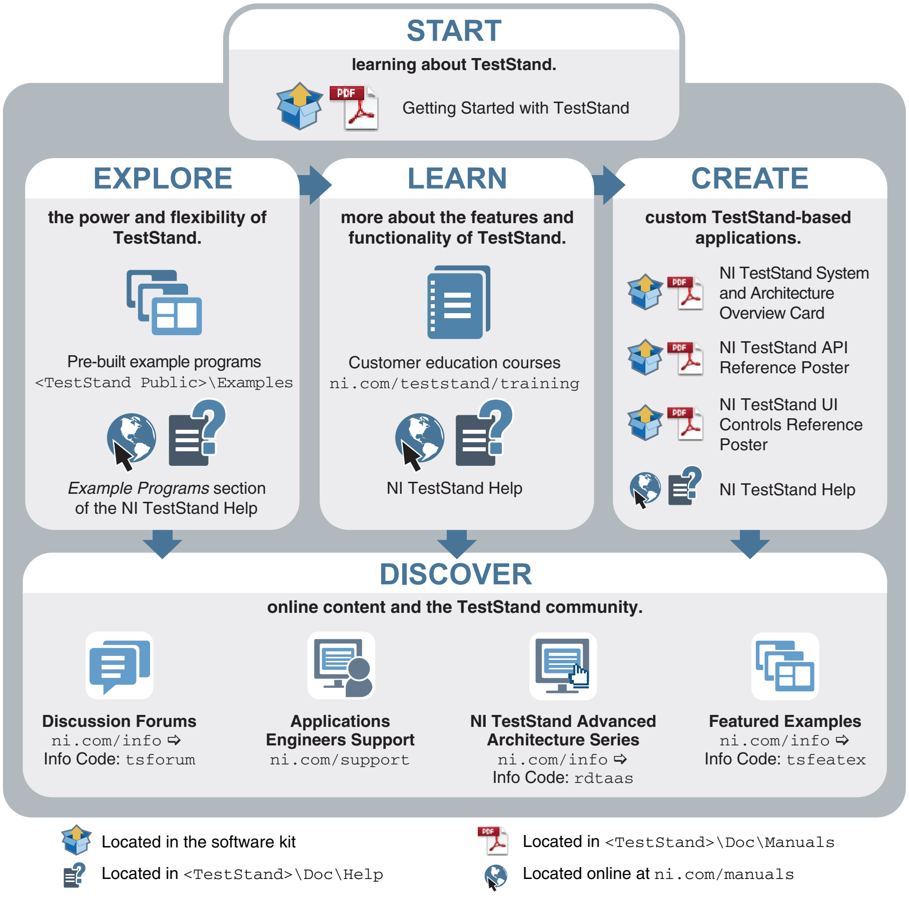
Use the *Guide to TestStand Documentation* topic in the *NI TestStand Help* to access all TestStand documentation in electronic format and to learn more about additional TestStand resources. You can access the *Guide to TestStand Documentation* topic in the following ways:

- Select **Help»Guide To Documentation** in the sequence editor.
- **(Windows 8.1/8)** Click the **NI Launcher** tile on the Start screen and select **TestStand»TestStand Documentation»TestStand Guide to Documentation**.
- **(Windows 7 or earlier)** Select **Start»All Programs»National Instruments»TestStand»TestStand Documentation»TestStand Guide to Documentation**.

The *NI TestStand System and Architecture Overview Card*, which you can access from the *Guide to TestStand Documentation* topic, includes a complete architectural diagram of TestStand, descriptions of the various system components, and diagrams that illustrate sequence file execution flow, `Execution` object structure, and sequence file structure. Use this card to familiarize yourself with TestStand concepts and refer to the card while you review this manual.

After you complete the tutorials in this manual, refer to the additional resources described in Figure 1-1 to learn more about developing a TestStand-based test system.

Figure 1-1. Additional TestStand Documentation and Resources

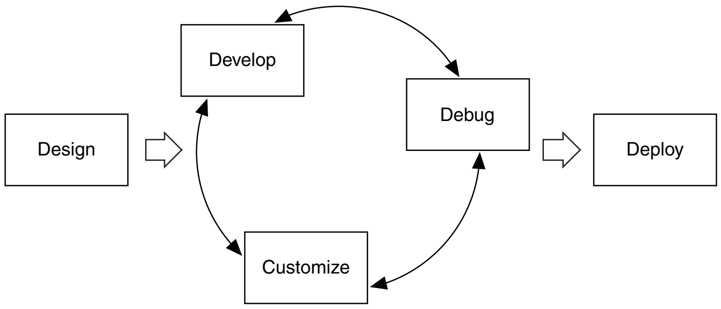


START
learning about TestStand.
Getting Started with TestStand
EXPLORE
the power and flexibility of TestStand.
Pre-built example programs
<TestStand Public>\Examples
Example Programs section of the NI TestStand Help
LEARN
more about the features and functionality of TestStand.
Customer education courses
ni.com/teststand/training
NI TestStand Help
CREATE
custom TestStand-based applications.
NI TestStand System and Architecture Overview Card
NI TestStand API Reference Poster
NI TestStand UI Controls Reference Poster
NI TestStand Help
DISCOVER
online content and the TestStand community.
Discussion Forums
ni.com/info ⇨
Info Code: tsforum
Applications Engineers Support
ni.com/support
NI TestStand Advanced Architecture Series
ni.com/info ⇨
Info Code: rdtaas
Featured Examples
ni.com/info ⇨
Info Code: tsfeatex
Located in the software kit
Located in <TestStand>\Doc\Help
Located in <TestStand>\Doc\Manuals
Located online at ni.com/manuals

Creating a TestStand-Based System

Creating a TestStand-based test solution is a process that includes designing the test system, developing test sequences for units under test (UUTs), customizing the TestStand framework, debugging the test sequences, and deploying the test system to test station computers, as Figure 1-2 shows.

Figure 1-2. TestStand System Development Life Cycle



National Instruments offers a variety of licenses for the different ways you can use TestStand in development and deployment applications. Refer to the *Licensing Options for TestStand Systems* section of this chapter for more information about each license.

Design

Learn about the components of the TestStand framework, learn how to use those features, and understand when to customize that behavior. Plan the system architecture of the proposed solution. Determine the development environments to use to develop user interfaces and code modules in the solution. Consider how to manage the development and deployment of the solution, including how to organize files under development, how to deploy the files, and how to debug and update systems you deploy.

Refer to the *Major Software Components of TestStand* section of this manual and to the *NI TestStand Help* for more information about the components and features of TestStand and how to customize those features. Refer to the *NI TestStand Advanced Architecture Series* for more advanced concept and architecture information. This series is a suite of documents TestStand architects and developers created to provide more detailed information for experienced TestStand users with complex projects. Visit `ni.com/info` and enter the Info Code `rdtaas` to locate the *NI TestStand Advanced Architecture Series*.

Note Before you start developing a test system, you must understand the concepts in the *Deploying TestStand Systems* portion of the *NI TestStand Help* so you can make appropriate design decisions that affect deployment as you complete development tasks.

Develop

Write test sequences for use on a test station computer. A test sequence is a series of steps that initialize instruments, perform complex tests, or change the flow of executions. Use the sequence editor or a custom sequence editor to write and edit test sequences.

Customize

Edit the default behavior of the TestStand framework depending on the needs of the application you create. You can customize reporting, database logging, process models, callback sequences, and user interfaces to create a unique, robust test solution for an application.

Debug

Ensure that the test sequences and any customized features execute correctly before you deploy the test system. Use the TestStand Sequence Analyzer in the sequence editor or the stand-alone sequence analyzer application during development or before deployment to find errors and enforce custom development guidelines you establish.

TestStand provides multiple features in the sequence editor or a custom sequence editor for debugging sequences, including tracing, breakpoints, conditional breakpoints, stepping through code, and watch expressions. The TestStand system development cycle is an iterative process, and you might have to debug an application multiple times.

Deploy

After you develop, customize, and debug a TestStand system, you can deploy the system to multiple test stations. The TestStand Deployment Utility simplifies the complex process of deploying a TestStand system by automating many of the steps involved, including collecting sequence files, code modules, configuration data for instruments, and support files for the test system. You can also use the deployment utility to create an installer or patch distributions.

Major Software Components of TestStand

The major software components of TestStand include the TestStand Engine, sequence editor, user interfaces, module adapters, process models, and deployment utility.

TestStand Engine

The TestStand Engine is a set of DLLs that exports an ActiveX Automation server API. The sequence editor and TestStand User Interfaces use the TestStand API, which you can call from any programming environment that supports access to ActiveX servers, including code modules you write in LabVIEW and LabWindows™/CVI™. The *NI TestStand API Reference Poster* and the *NI TestStand User Interface Controls Reference Poster* include an overview of the TestStand API. The *NI TestStand Help* includes the detailed documentation for the TestStand API.

TestStand Sequence Editor

The sequence editor is a development environment in which you create, edit, execute, and debug sequences and the tests sequences call. Use the sequence editor to access all features, such as step types and process models. Refer to the *Process Models* section of this chapter for more information about process models.

You can debug a sequence using the following techniques, similar to how you debug in application development environments (ADEs) such as LabVIEW, LabWindows/CVI (ANSI), and Microsoft Visual Studio:

- Setting breakpoints
- Stepping into, out of, or over steps
- Tracing through program executions
- Displaying variables
- Monitoring variables, expressions, and output messages during executions
- Performing static analysis of sequence files to locate errors and enforce coding guidelines

In the sequence editor, you can start multiple concurrent executions, execute multiple instances of the same sequence, or execute different sequences at the same time. Each execution instance opens an Execution window. In Trace Mode, the Execution window shows the steps in the currently executing sequence. If the execution suspends, the Execution window shows the next step to execute and provides debugging options.

In the sequence editor, you can fully customize the pane and tab layout to optimize development and debugging tasks. You can also customize the menus, toolbars, and keyboard shortcuts.

User Interfaces

A TestStand User Interface is an application you deploy to a development system or a production station to provide a custom GUI for executing, debugging, or editing sequences. Simple user interfaces might only support running sequences, and custom sequence editors might support editing, running, and debugging sequences.

TestStand includes separate user interface applications developed in LabVIEW, LabWindows/CVI, Microsoft Visual Basic .NET, C#, and C++ (MFC). Because TestStand also includes the source code for each user interface, you can fully customize the user interfaces. You can create a custom user interface using any programming language that can host ActiveX controls or control ActiveX Automation servers.

With the user interfaces in Editor Mode, you can modify sequences and display sequence variables, sequence parameters, step properties, and so on. With the user interfaces in Operator Mode, you can start multiple concurrent executions, set breakpoints, and step through sequences.

Module Adapters

The TestStand Engine uses module adapters to invoke code modules that sequences call. A code module is a program module from an ADE or programming language and can contain one or more functions that perform a specific test or other action. Module adapters load and call code modules, pass parameters to code modules, and return values and status from code modules. The module adapters support the following types of code modules:

- LabVIEW VIs
- LabWindows/CVI functions in DLLs you create in LabWindows/CVI or other compilers
- C/C++ functions in DLLs
- .NET assemblies
- ActiveX Automation servers
- HTBasic subroutines

Adapters specific to an ADE can open the ADE, create source code for a new code module in the ADE, and display the source for an existing code module in the ADE. The adapters support stepping into the source code in the ADE while you execute the step from the TestStand Sequence Editor or a TestStand User Interface.

TestStand includes the following module adapters:

- **LabVIEW Adapter**—Calls LabVIEW VIs with a variety of connector panes. The VIs can exist in LLBs or in LabVIEW packed project libraries. You can also call VIs in the context of a LabVIEW project or call LabVIEW class member VIs.
- **LabWindows/CVI Adapter**—Calls C functions in a DLL with a variety of parameter types.
- **C/C++ DLL Adapter**—Calls C/C++ functions and static C++ class methods in a DLL with a variety of parameter types. You can call global static methods or static class methods in C++ DLLs. You can create the DLL code module with LabWindows/CVI, Visual Studio, or any other environment that can create a C/C++ DLL, including LabVIEW-built shared libraries.
- **.NET Adapter**—Calls .NET assemblies written in any .NET-compliant language, such as C# or Microsoft Visual Basic .NET.
- **ActiveX/COM Adapter**—Creates ActiveX/COM objects, calls methods, and accesses properties of those objects. When you create an object, you can assign the object reference to a variable or property for later use in other ActiveX/COM Adapter steps.
- **HTBasic Adapter**—Calls HTBasic subroutines without passing parameters directly to a subroutine. TestStand provides a library of CSUB routines that use the TestStand API to access TestStand variables and properties from an HTBasic subroutine.

Note **(64-bit TestStand)** The HTBasic Adapter is not supported.

- **Sequence Adapter**—Calls a subsequence in the current sequence file, in another sequence file, or a sequence file on a remote system. You can also make recursive sequence calls.

Refer to Chapter 11, *Calling LabVIEW VIs*, and to Chapter 12, *Calling LabWindows/CVI Code Modules*, for more information about using LabVIEW and LabWindows/CVI, respectively, with TestStand. Refer to the *NI TestStand Help* for more information about using the LabVIEW and LabWindows/CVI Adapters.

Process Models

Testing a UUT requires more than just executing a set of tests. Usually, the test system must perform a series of operations before and after it executes the sequence that performs the tests. Common operations that define the testing process include identifying the UUT, notifying the operator of pass/fail status, logging results, and generating a report. The set of operations and the flow of execution is called a process model. A TestStand process model is a sequence file you can use to define standard testing operations so you do not have to re-implement the same operations in every sequence file you write.

TestStand includes predefined Sequential, Parallel, and Batch models you can modify or replace. Use the Sequential model to run a test sequence on one UUT at a time. Use the Parallel and Batch models to run the same test sequence on multiple UUTs simultaneously.

You can modify an existing TestStand process model or you can create a custom process model. The ability to modify a process model is essential because the testing process can vary depending on production lines, production sites, or company systems and practices. You can edit a process model in the same way you edit other sequence files. You can also use client sequence files to customize various model operations by overriding the callback sequences process models define.

Entry Points

A process model defines a set of entry points, and each entry point is a sequence in the process model file that invokes a test sequence file. Defining multiple entry points in a process model gives the test station operator different ways to invoke a Main sequence or configure the process model.

Execution entry points in a process model provide different ways for the test station operator to invoke a Main sequence. Execution entry points handle common operations, such as UUT identification and report generation. For example, the default Sequential process model provides the Test UUTs and Single Pass Execution entry points. The Test UUTs Execution entry point initiates a loop that repeatedly identifies and tests UUTs. The Single Pass Execution entry point tests a single UUT without identifying it.

Configuration entry points provide an interface for configuring the process model, typically through a GUI. For example, the default Batch model provides the Configure Model Options entry point. This entry point creates the **Configure** menu item.

TestStand Deployment Utility

Use the TestStand Deployment Utility to create a deployable image or a patch deployment of a TestStand system and an optional installer. The deployable image can contain sequence files, code modules, process models and supporting files, user interface applications, configuration files, and step types and supporting files the TestStand system uses. The installer can contain all files from a deployable image or contain only a subset of files to create a patch for a previously deployed image.

You can also use the deployment utility to include the TestStand Engine and supporting files, LabVIEW and LabWindows/CVI Run-Time Engines, and hardware drivers in the installer you create. The installer that the deployment utility creates can also register ActiveX servers, replace existing files on the target computer, and create program shortcuts. You can configure the deployment utility to remove VI block diagrams or to lock the VIs you deploy.

Licensing Options for TestStand Systems

National Instruments offers a variety of licenses for the different ways you can use TestStand in development and deployment applications. Refer to the *NI TestStand Help* or to the *Activating TestStand Licenses* section of the *NI TestStand Release Notes* for more information about the differences among the TestStand licensing options.

- **Designing, developing, and debugging sequences**—You must have one of the following licenses to write sequences:
  - TestStand Development System License
  - TestStand Custom Sequence Editor License

  You typically use the TestStand Debug Deployment Environment License to debug and fix deployed test systems.
- **Developing a user interface**—You must have the following license to develop a custom user interface, including a custom sequence editor:
  - TestStand Development System License
- **Creating a deployment**—You must have one of the following licenses to create a deployment:
  - TestStand Development System License
  - TestStand Custom Sequence Editor License
- **Deploying a TestStand system**—You must have one of the following licenses on the computer to which you deploy a test system, and each test station for a deployed test system must have its own copy of the license:
  - TestStand Base Deployment Engine License
  - TestStand Debug Deployment Environment License
- **Deploying a custom sequence editor**—You must have one of the following licenses on the computer to which you deploy a test system that includes a custom sequence editor, and each deployed test system must have its own copy of the license:
  - TestStand Custom Sequence Editor License
  - TestStand Development System License

You cannot activate and deactivate the TestStand Debug Deployment Environment License and reuse it on multiple computers. If you need to use a single debug license across multiple computers, contact National Instruments for more information about the Concurrent TestStand Debug Deployment Environment License.

You can use a TestStand Development System License on a computer to which you have deployed a test system to perform any operations you would perform on a development computer. You can also run TestStand on a test station computer in Evaluation Mode until the software evaluation period expires.

2

Loading and Running Sequences

The TestStand Sequence Editor includes menus, toolbars, windows, and panes you use to load workspace files and edit and run sequence files. Click the **Help Topic (F1)** button, as shown in the following figure and located on the toolbar, to launch the *NI TestStand Help* for specific information about the active window, tab, or pane in the sequence editor.

Refer to the *NI TestStand Help* for more information about all other features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Starting TestStand

Complete the following steps to launch the sequence editor.

1. **(Windows 8.1/8)** Click the **NI Launcher** tile on the Start screen and select **TestStand»TestStand Sequence Editor**.

   (Windows 7 or earlier) Select **Start»All Programs»National Instruments»TestStand»TestStand Sequence Editor**.

   The sequence editor launches the main window and the Login dialog box.
2. Use the default user name, `administrator`, in the User Name ring control. Leave the Password field empty.
3. Click **OK**.

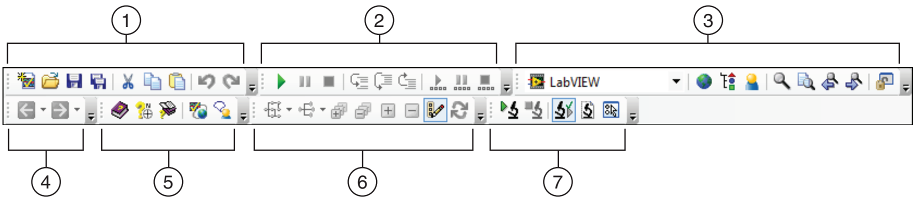
Menu Bar

By default, the menu bar contains the File, Edit, View, Execute, Debug, Configure, Source Control, Tools, Window, and Help menus. The menus are fully customizable. You can create menus with any sequence editor commands you want. Browse the menus in the sequence editor to familiarize yourself with the contents of each menu.

Toolbars

By default, the sequence editor contains the following toolbars with shortcuts to commonly used selections from the menu bar, as shown in Figure 2-1.

Figure 2-1. Sequence Editor Toolbars



1	Standard Toolbar	5	Help Toolbar
2	Debug Toolbar	6	Sequence Hierarchy Toolbar
3	Environment Toolbar	7	Sequence Analyzer Toolbar
4	Navigation Toolbar		

- **Standard Toolbar**—Contains buttons for creating, loading, and saving sequence files, and for cutting, copying, and pasting. This toolbar also includes the **Undo** and **Redo** buttons.
- **Debug Toolbar**—Contains buttons for executing a sequence, stepping into, stepping over, stepping out of, terminating, and suspending executions.
- **Environment Toolbar**—Contains the following items:
  - Adapter ring control
  - Buttons for opening station globals, type palettes, and the TestStand User Manager
  - Buttons for performing search and replace operations
  - Button for unlocking and locking the user interface (UI) configuration to enable or disable customizing various aspects of the UI in the development environment
- **Navigation Toolbar**—Contains the **Back** and **Forward** buttons, which you use to show the previous or next view in the history list of the sequence file or Sequence Hierarchy window.
- **Help Toolbar**—Contains buttons to launch the *NI TestStand Help*, the TestStand support page of `ni.com`, and the TestStand Discussion Forum on `ni.com`.
- **Sequence Hierarchy Toolbar**—Contains buttons for using the Sequence Hierarchy window, which displays a graph of the sequence call and callback sequence relationships among sequence files and sequences to more easily visualize, navigate, and maintain test sequences.
- **Sequence Analyzer Toolbar**—Contains buttons for analyzing a sequence file or a TestStand Sequence Analyzer project to detect errors and for configuring the sequence analyzer options.

The toolbars are fully customizable. You can create toolbars with buttons for any environment commands you want.

Windows and Panes

The sequence editor contains tabbed windows and panes you can float, dock, resize, and hide. You can customize the pane and tab layout to optimize development and debugging tasks. You can reset the user interface configuration state of the panes in the sequence editor by selecting **View»Reset UI Configuration**, which resets to a default configuration all modifications previously made to menus, toolbars, and the docking state of panes.

Note When you launch the sequence editor for the first time or when you reset the sequence editor configuration, the layout of the panes defaults to the Small Screen Example configuration or the Large Screen Example configuration, depending on the screen resolution. TestStand considers screens with a resolution of 1280 × 1024 or greater to be large.

Select **Configure»Sequence Editor Options** to launch the Sequence Editor Options dialog box and select a sequence editor configuration. On the UI Configuration tab, select an example configuration in the **Saved Configurations** control and click the **Load Selected** button. Click **OK** to close the Sequence Editor Options dialog box.

The Insertion Palette pane in the TestStand Sequence Editor displays items you can insert into a sequence file. The pane contains a Step Types list and a Templates list. The Step Types list displays the step type menu, and the Templates list displays user-defined template sequences, steps, and variables.

Status Bar

The status bar displays the following information for the sequence editor:

- User name of the current user.
- Name of the process model file the current sequence uses. You can double-click in this area to open that process model file.
- Number of steps you have selected.
- Total number of steps in the active sequence.
- File path of the active report after execution.

Organizing a TestStand System with Workspace and Project Files

Create a workspace to organize and access development files. Use workspaces early in development so you can easily keep track of files while you are developing. A workspace file (`.tsw`) contains references to any number of TestStand project files. A project file (`.tpj`) contains references to any number of other files of any type.

Use project files to organize related files and directories of files in the test system. You can insert any number of files into a project. For example, you can use a project file to organize a sequence file, code module source files, and other supporting files.

A workspace file opens on the Workspace pane, which displays the content of a workspace file. To open any file from the Workspace pane, double-click the file. To organize the files, use the options in the Edit menu to cut, copy, paste, and delete files in projects or project files in the workspace. You can also drag and drop files on the Workspace pane.

You can open only one workspace file at a time. If you have a workspace file open and you try to open or create another, TestStand prompts you to save the workspace file before opening or creating another file.

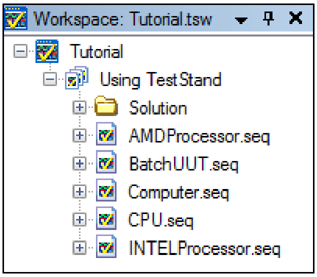
You must use workspace files to use the source code control (SCC) system features in TestStand. You can also use workspace files to deploy test systems with the TestStand Deployment Utility.

Loading a Workspace File

Complete the following steps to load and view a workspace file.

1. Select **File»Open File** and navigate to the `<TestStand Public>\Tutorial` directory.
2. Select `Tutorial.tsw` and click **Open**.
3. Expand the **Using TestStand** project, which contains all the sequence files used in the tutorials in this manual, including the `Solution` directory, as shown in Figure 2-2.

Figure 2-2. Workspace Pane with the Using TestStand Project Expanded



Workspace: Tutorial.tsw

```
Tutorial
  Using TestStand
    Solution
    AMDProcessor.seq
    BatchUUT.seq
    Computer.seq
    CPU.seq
    INTELProcessor.seq
```

You can leave the workspace file open on the Workspace pane so you can directly open the sequence files for the tutorials in this manual, or you can close the Workspace pane and use the options in the File menu to open the sequence files. If you want to close the Workspace pane, select **File»Close Workspace File** and click **No** if TestStand prompts you to save the changes.

Creating Test Sequences and Sequence Files

A sequence consists of a series of steps. A step can perform many actions, such as initializing an instrument, performing a complex test, or controlling the flow of execution in a sequence.

A sequence file can contain one or more sequences. Sequence files can also contain global variables, which all sequences in the sequence file can access.The sequence editor color-codes sequences by type, such as Execution entry points, Configuration entry points, process model callback sequences, subsequences, and engine callbacks.

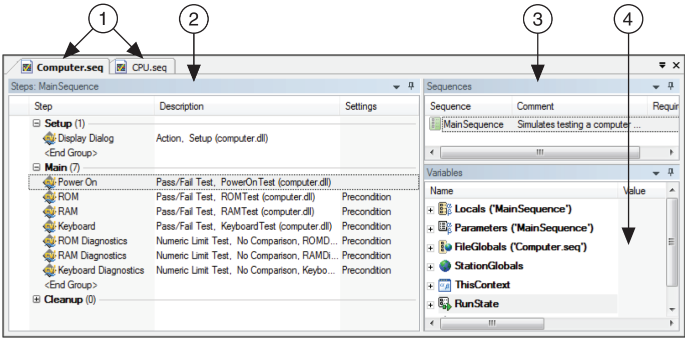
The Setup, Main, and Cleanup groups of the Steps pane display a list of the steps in the step group.TestStand executes the steps in the Setup step group first, the Main step group second, and the Cleanup step group last. The Setup step group typically contains steps that initialize instruments, fixtures, or a UUT. The Main step group typically contains the bulk of the steps in a sequence, including the steps that test the UUT. The Cleanup step group typically contains steps that power down or restore the initial state of instruments, fixtures, and the UUT.

A sequence can have any number of parameters and local variables. Use parameters to pass data to a sequence when you call the sequence as a subsequence. Use local variables for storing data relevant to the execution of the sequence, storing any other data needed only in the current sequence, maintaining counts, or holding intermediate values.

In the sequence editor, the Variables pane displays all the variables and properties the selected sequence has access to at run time. When you execute a sequence, the Variables pane displays the sequence context for the sequence currently selected on the Call Stack pane. The sequence context contains all the variables and properties accessible in the current execution. Use the Variables pane to examine and modify the values of these variables and properties when an execution is suspended.

The sequence editor opens each sequence file in a separate Sequence File window, which includes the Sequences pane, the Steps pane, and the Variables pane, as shown in Figure 2-1. The Sequences pane lists the name, comment, and requirements values for all sequences in the sequence file. Use the Sequences pane to select the active sequence to display on the Steps pane, to insert new sequences, and to delete existing sequences from a sequence file.

Figure 2-3. Sequence File Window



Step	Description	Settings
Setup (1)		
Display Dialog	Action, Setup (computer.dll)	
<End Group>		
Main (7)		
Power On	Pass/Fail Test, PowerOnTest (computer.dll)	
ROM	Pass/Fail Test, ROMTest (computer.dll)	Precondition
RAM	Pass/Fail Test, RAMTest (computer.dll)	Precondition
Keyboard	Pass/Fail Test, KeyboardTest (computer.dll)	Precondition
ROM Diagnostics	Numeric Limit Test, No Comparison, ROMD...	Precondition
RAM Diagnostics	Numeric Limit Test, No Comparison, RAMDi...	Precondition
Keyboard Diagnostics	Numeric Limit Test, No Comparison, Keybo...	Precondition
<End Group>		
Cleanup (0)		

Sequence	Comment	Requir
MainSequence	Simulates testing a computer ...	

Name	Value
Locals ('MainSequence')	
Parameters ('MainSequence')	
FileGlobals ('Computer.seq')	
StationGlobals	
ThisContext	
RunState	

1	Sequence File Windows	2	Steps Pane	3	Sequences Pane	4	Variables Pane

Loading a Sequence File

Complete the following steps to load and view a sequence file.

1. Select **File»Open File** and navigate to the `<TestStand Public>\Tutorial` directory.
2. Select `Computer.seq` and click **Open**. The `Computer.seq` sequence file is a simulated test of a computer in which you can designate various hardware components to fail. The sequence runs tests implemented as functions in a DLL. TestStand includes examples of DLLs written in LabVIEW, .NET, and LabWindows/CVI.

Note If you left the `Tutorial.tsw` workspace file open from the *Loading a Workspace File* section of this chapter, you can open `Computer.seq` from the Workspace pane by double-clicking `Computer.seq` in the list of files.

3. Select **MainSequence** on the Sequences pane.
4. Browse the contents of each pane in the Sequence File window. Ensure that the Sequences pane and the Steps pane are visible when you finish.

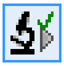
Executing a Sequence

You can execute a sequence directly or execute a sequence using a process model Execution entry point. The process model entry point sequence contains a series of steps that specify the high-level flow of an execution.

The TestStand Sequence Editor launches a separate Execution window for each execution you start using the Execute menu. Use the Execution window to view, or trace, steps as they execute, to monitor the values of variables and properties, and to examine test reports when the execution completes.

In Trace Mode, the Execution window shows the steps in the currently executing sequence and indicates the currently executing step. If the execution suspends, the Execution window shows the next step to execute and provides debugging options.

Before the execution begins, by default, the TestStand Sequence Analyzer analyzes the active sequence file and detects the most common situations that can cause run-time failures. The sequence analyzer prompts you to resolve the reported errors before you execute the sequence file. Carefully review any errors the sequence analyzer returns to prevent run-time failures. If you want to disable the sequence analyzer, click the **Toggle Analyze File Before Executing** button, as shown in the following figure and located on the Sequence Analyzer toolbar.



Executing a Sequence Directly

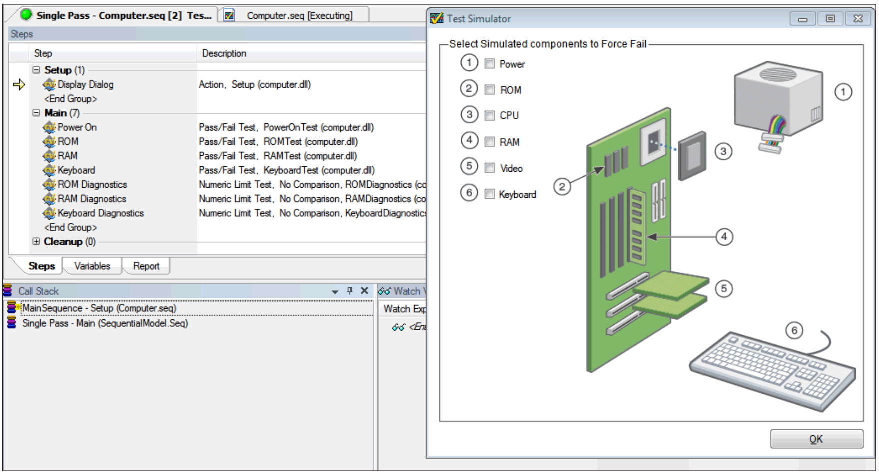
Complete the following steps to run `MainSequence` in the `Computer.seq` sequence file directly.

1. Select **Configure»Station Options** to launch the Station Options dialog box.
   The Execution tab contains options for configuring how an execution behaves. By default, the Enable Tracing option is enabled to specify that the sequence editor displays the progress of an execution with a yellow arrow icon that appears to the left of the currently executing step. This icon is called the execution pointer, as shown in the following figure.

   The Execution tab also contains a Speed slider control that specifies the tracing speed of the execution. Use this option to slow down the tracing speed so you can observe each step as it executes. By default, the tracing speed is set to fast.
2. Adjust the **Speed** slider control to slow. Click **OK** to close the Station Options dialog box.
3. Select **Execute»Run MainSequence**.

   The sequence editor opens an Execution window to show the sequence as it executes. The first step in the Setup step group of `MainSequence` launches the Test Simulator dialog box, as shown in Figure 2-4. The Test Simulator dialog box prompts you to designate the computer component, if any, you want to fail during the execution.

Figure 2-4. MainSequence Execution Window



Step	Description
Setup (1)	
Display Dialog	Action, Setup (computer.dll)
<End Group>	
Main (7)	
Power On	Pass/Fail Test, PowerOnTest (computer.dll)
ROM	Pass/Fail Test, ROMTest (computer.dll)
RAM	Pass/Fail Test, RAMTest (computer.dll)
Keyboard	Pass/Fail Test, KeyboardTest (computer.dll)
ROM Diagnostics	Numeric Limit Test, No Comparison, ROMDiagnostics (co
RAM Diagnostics	Numeric Limit Test, No Comparison, RAMDiagnostics (co
Keyboard Diagnostics	Numeric Limit Test, No Comparison, KeyboardDiagnostic
<End Group>	
Cleanup (0)	

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

Select Simulated components to Force Fail
1 Power
2 ROM
3 CPU
4 RAM
5 Video
6 Keyboard

4. Enable the **RAM** test option.
5. Click **Done**.

6. Observe the Execution window as it traces through the steps TestStand runs. After the execution completes, the Execution window dims, and the Status column of the `RAM` test contains the value `Failed`.

Note If the tracing speed is too slow or fast, adjust the **Speed** slider control on the Execution tab of the Station Options dialog box.

7. Right-click the Execution window tab and select **Close** from the context menu to close the Execution window.

Executing a Sequence Using the Sequential Process Model

In addition to executing sequences directly, you can execute sequences using an Execution entry point, which is a sequence in a process model file that invokes a test sequence file, typically by calling the MainSequence callback in the client sequence file. Executing an Execution entry point performs a series of operations before and after calling `MainSequence` of the sequence file. Common operations of the process model include identifying the UUT, notifying the operator of pass/fail status, logging results, and generating reports.

The Sequential process model includes the following Execution entry points:

- **Test UUTs Execution Entry Point**—The Test UUTs Execution entry point initiates a loop that repeatedly identifies and tests UUTs.
- **Single Pass Execution Entry Point**—The Single Pass Execution entry point tests a single UUT without identifying it. Use the Single Pass Execution entry point when you want to debug tests or determine whether the sequence execution proceeds as you intended.

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

1. Launch the Station Options dialog box.
2. Click the **Model** tab, verify that `SequentialModel.seq` is selected from the Station Model ring control to select the Sequential model as the default process model, and click **OK**.
3. Select **Execute»Test UUTs**. Before executing the steps in `MainSequence`, the process model sequence launches a UUT Information dialog box that prompts you for a serial number.
4. Enter any serial number and click **OK**.
5. Use the options in the Test Simulator dialog box to select any test other than the Video or CPU tests to fail. You can also allow all the tests of the UUT to pass.
6. Click **Done**. Observe the Execution window as the sequence executes. After completing the steps in `MainSequence`, the process model displays a banner that indicates the result of the UUT.

7. Click **OK** to close the UUT Result banner. TestStand generates a report, but does not display the report until you finish testing all the UUTs. TestStand launches the UUT Information dialog box again.
8. Repeat steps 4 through 7 using several different serial numbers.
9. Click **Stop** in the UUT Information dialog box to stop the loop and complete the execution. After the execution completes, TestStand displays the report on the Report pane of the Execution window for all the tested UUTs.
10. Review the test report, which includes the results for each UUT, and select **Window» Close All Windows** to close all the windows in the sequence editor.

Note You must have a minimum supported version of Microsoft Internet Explorer or later to view TestStand reports. Refer to the *NI TestStand Release Notes* for more information about supported versions of Internet Explorer.

Executing a Sequence Using the Batch Process Model

Use the Batch model to control a set of test sockets that test multiple UUTs as a group. For example, if you have a set of circuit boards attached to a common carrier, use the Batch model to ensure that you start and finish testing all boards at the same time. With the synchronization features of the Batch model, you can direct a step that applies to the batch as a whole to run only once per batch instead of once for each UUT. You can also specify whether certain steps or groups of steps cannot run on more than one UUT at a time or whether certain steps must run on all UUTs at the same time. The Batch model generates batch reports that summarize the test results for the UUTs in the batch.

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

1. Open `<TestStand Public>\Tutorial\BatchUUT.seq` and examine the steps and comments in the sequence file to familiarize yourself with the functionality of the sequence.

Note You do not need to change the default process model in the sequence editor for this tutorial. The `BatchUUT.seq` sequence file always uses the Batch model, regardless of the default process model of the sequence editor. Use the Advanced tab of the Sequence File Properties dialog box to specify the process model a sequence file uses.

2. Select **Configure»Model Options** to launch the Model Options dialog box.
3. In the Multiple UUT Settings section, change **Number of Test Sockets** to `2` and enable the **Tile Execution Windows** option. The Number of Test Sockets control specifies the number of UUTs to test in the batch.
4. In the Batch Settings section, select **Don’t Synchronize** from the Default Batch Synchronization ring control and click **OK**.
5. Select **Execute»Test UUTs**. Before executing the steps in `MainSequence`, the process model sequence launches the UUT Information dialog box for the Batch model, which

prompts you for a batch serial number and UUT serial numbers for each test socket. You can also disable test sockets in the UUT Information dialog box.

6. Enter any batch serial number and UUT serial numbers. Click **Go**. Review the information in the Batch Model Example dialog box, then click **OK**. TestStand launches several different front panels to indicate the progress of the executions. Click **OK** to continue.

   After completing the steps in `MainSequence`, the process model launches the Batch Results dialog box, which displays a banner that indicates the result of the UUTs. You can view the batch report and individual UUT reports or start the next batch.
7. Click the **View Batch Report** button. From the pop-up list, you can select Entire File to show all tested batches or Current Only to show the most recently tested batch.
8. Select **Current Only** from the pop-up list. TestStand launches the default application associated with the file extension of the report, such as Internet Explorer for Automatic Test Markup Language (ATML), XML, and HTML reports and Microsoft Notepad for text reports.

Note Enable Internet Explorer to run scripts if the browser launches a prompt.

   To use another application as an external report viewer, select **Configure»External Viewers** and follow the instructions in the Configure External Viewers dialog box. Refer to the *Using External Report Viewers* section of Chapter 10, *Customizing Reports*, for more information about using external viewers.
9. Close Internet Explorer and click **Next Batch** in the Batch Results dialog box.
10. Repeat steps 6 through 9 for several different batches.
11. Click **Stop** in the UUT Information dialog box to complete the execution. After the execution completes, TestStand displays test reports on the Report pane of the Execution window for all batches and UUTs.
12. Examine the reports that include results for each batch and UUT.
13. Close all the windows in the sequence editor.

3 Editing Steps in a Sequence

You can add a step to a sequence, configure the step to call a code module or subsequence, and configure the properties of a step.

The Insertion Palette contains a set of predefined step types you can add to sequences. Step types define the standard behavior and a set of step properties for each step of that type. All steps of the same type have the same properties, but the values of the properties can differ.

When you build sequence files, you can also use the Templates list in the Insertion Palette. Use the Templates list to hold copies of steps, variables, and sequences you reuse during the development of sequence files. For example, you can add a step that calls a specific LabVIEW VI you typically use or a sequence that contains common setup steps, cleanup steps, and local variables.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Adding a New Step

Complete the following steps to add a Pass/Fail Test step to a sequence and configure the step to call a function in a LabWindows/CVI DLL code module by specifying the module adapter to use. TestStand does not need the source code to invoke a DLL code module. Instead, TestStand uses module adapters to determine the type of code module, how to call the code module, and how to pass parameters to the code module.

1. Open `<TestStand Public>\Tutorial\Computer.seq`.
2. Select **File»Save *<filename>* As** and save the sequence file as `Computer2.seq` in the `<TestStand Public>\Tutorial` directory.

Note When you must save a file, this manual specifies the suggested name. If other users will use the tutorial files on the same computer, save the files with unique filenames.

3. Click the **LabWindows/CVI** icon, as shown in the following figure and located at the top of the Insertion Palette, to specify the module adapter the step uses.

   You can also select adapters from the Adapter ring control on the Environment toolbar. The adapter you select applies only to the step types that can use the module adapter.

   Note If you left the `Tutorial.tsw` workspace open in Chapter 2, *Loading and Running Sequences*, you might need to click the **Insertion Palette** tab, which uses the icon shown in the following figure and is located at the bottom of the Workspace pane, to show the Insertion Palette.

   When you insert a step in a sequence, TestStand configures the step to use the adapter you selected from the Insertion Palette. The icon for the step reflects the adapter you selected. When you select <None> as the adapter and then insert a step, the new step does not call a code module. Use the General panel on the Properties tab of the Step Settings pane to change the associated adapter after you insert the step.
4. On the Insertion Palette, select **Tests»Pass/Fail Test** and drag the step below the RAM step on the Steps pane to add a Pass/Fail Test step. By default, the name of the new step is Pass/Fail Test.

   Note You can also create steps by right-clicking the Steps pane, selecting **Insert Step** from the context menu, and selecting the type of step you want to insert.

   Use a Pass/Fail Test step to call a code module that returns a pass/fail determination. After the code module executes, the Pass/Fail Test step evaluates a Boolean expression to determine whether the step passed or failed.
5. Rename the new step `Video Test` by selecting the step on the Steps pane and pressing the <F2> key.
6. Save the changes. Leave the sequence file open for the next tutorial.

Specifying the Code Module

Complete the following steps to specify the code module the step executes and to specify the parameters for a function in the code module for the `Computer2.seq` sequence file you created in the previous tutorial.

1. Select the **Video Test** step and click the **Module** tab of the Step Settings pane.
2. Click the **Browse** button located to the right of the Module control, select `<TestStand Public>\Tutorial\computer.dll`, and click **Open**. When you select a DLL, TestStand reads the type library or exported information of the DLL and lists the functions TestStand can call in the Function ring control.
3. Select **VideoTest** from the Function ring control. TestStand uses the prototype information stored in the type library or the exported information of the DLL to populate the Parameters Table.
4. In the Value Expression column of the **result** parameter, enter `Step.Result.PassFail`.

   When TestStand returns from calling the `VideoTest` function during execution, TestStand assigns the value from the **result** parameter to the `Step.Result.PassFail` property of the step.
5. Save the changes. Leave the sequence file open for the next tutorial.

Refer to Chapter 11, *Calling LabVIEW VIs*, for more information about calling LabVIEW VIs from TestStand. Refer to Chapter 12, *Calling LabWindows/CVI Code Modules*, for more information about calling LabWindows/CVI code modules from TestStand.

Configuring Step Properties

Note Before you begin this tutorial, select **Configure»Station Options** to launch the Station Options dialog box, click the **Execution** tab, and confirm that the Enable Tracing and Allow Tracing into Setup/Cleanup options are enabled. Click **OK** to close the Station Options dialog box.

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

- When to load the code module for the step
- When to execute the step
- What information TestStand examines to determine the status of the step
- Whether TestStand executes the step in a loop
- What conditional actions occur after a step executes

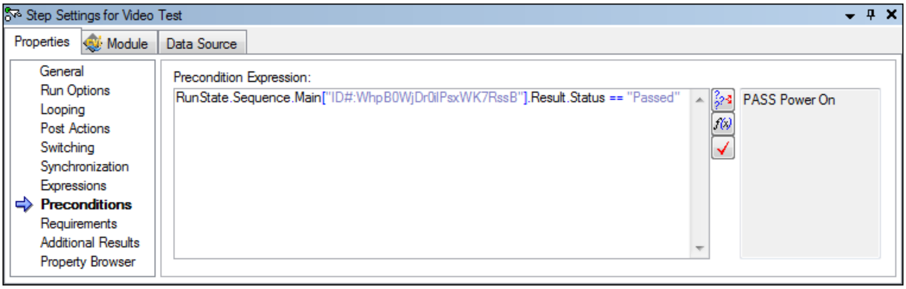
Steps can also contain additional properties the step type defines. Use the Properties tab of the Step Settings pane to examine and modify the values of properties of a step.

Complete the following steps to examine and modify step properties in the `Computer2.seq` sequence file you created in the previous tutorial.

1. Select the **Video Test** step on the Steps pane and click the **Properties** tab of the Step Settings pane.
2. Click **Preconditions** on the Properties tab to show the Preconditions panel. A precondition is a set of conditions for a step that must evaluate to `True` for TestStand to execute the step during the normal flow of execution in a sequence.
3. Complete the following steps to define a precondition so the Video Test step executes only if the Power On step passes.
   a. Click the **Precondition Builder** button, as shown in the following figure and located to the right of the Precondition Expression control on the Preconditions panel, to launch the Precondition Builder dialog box.

   b. In the Insert Step Status section, select the **Power On** step from the list of step names for the Main step group and click the **Insert Step Pass** button. The Conditions control now contains the string `PASS Power On`, which indicates that the step executes only if the Power On step passes.
   c. Click **OK** to close the Precondition Builder dialog box and confirm that the Preconditions panel matches the settings shown in Figure 3-1.

Figure 3-1. Preconditions Pane


Step Settings for Video Test

Properties | Module | Data Source

General
Run Options
Looping
Post Actions
Switching
Synchronization
Expressions
Preconditions
Requirements
Additional Results
Property Browser

Precondition Expression:

RunState.Sequence.Main["ID#:WhpB0WjDr0ilPsxWK7RssB"].Result.Status == "Passed"

PASS Power On

4. Click **Post Actions** on the Properties tab of the Step Settings pane to show the Post Actions panel, on which you can specify what type of action occurs after the step executes. You can make the action conditional on the pass/fail status of the step or on any custom condition expression.
5. Select **Terminate execution** from the On Fail ring control.
6. Click **Looping** on the Properties tab of the Step Settings pane to show the Looping panel, on which you can configure an individual step to run repeatedly in a loop when the step executes. Use the Loop Type ring control to select the type of looping for the step.

TestStand determines the final status of the step based on the number of passes, failures, or loop iterations that occur.

7. On the Looping panel, enter the following values into the corresponding controls. When you change the value of a property, TestStand shows the new value in bold to indicate that it differs from the default value.
   - **Loop Type**—Fixed number of loops
   - **Number of Loops**—`10`
   - **Loop result is Fail if**—`<80%`

   Using these settings, TestStand executes the Video Test step 10 times and sets the overall status for the step to `Failed` if fewer than 8 of the 10 iterations pass.
8. Confirm that the Settings column on the Steps pane of the Sequence File window shows that the Video Test step contains Loop, Precondition, and Post Action settings. Use the tooltip in the Settings column to verify the values for each setting.
9. Save the changes and select **Execute»Single Pass**.
10. Select the **Video** test to fail and click **Done**. The sequence immediately terminates after calling the Video Test step 10 times in a loop. TestStand records the result of each loop iteration in the report.
11. Close the Execution window.

Calling Subsequences

Use the Sequence Call step to call another sequence in the current sequence file or in another sequence file.

Complete the following steps to add a Sequence Call step to the `Computer2.seq` sequence file you created in the previous tutorial.

1. Insert a Sequence Call step after the Power On step on the Steps pane and rename the step `CPU Test`.
2. On the Module tab of the Step Settings pane for the CPU Test step, click the **Browse** button located to the right of the File Pathname control and select `<TestStand Public>\Tutorial\CPU.seq` to specify the sequence the step invokes.
3. Select **MainSequence** from the Sequence ring control on the Module tab of the Step Settings pane.
4. Save the changes and double-click the **CPU Test** step to open the sequence and show the `MainSequence` of `CPU.seq` in a new Sequence File window.
5. Select **Execute»Run MainSequence**. Examine the execution of `CPU.seq` so you can recognize it later when you execute the `Computer2.seq` sequence file that calls `CPU.seq`.
6. Close the Execution window and the `CPU.seq` Sequence File window.
7. In the `Computer2.seq` Sequence File window, select **Execute»Single Pass**.

8. Select a test to fail and click **Done**. After the sequence executes, review the test report. TestStand logs the results of the steps in the subsequence in addition to the steps from the parent sequence.
9. Close the Execution window. Leave the `Computer2.seq` sequence file open for the next tutorial.

Using Step Templates

The Insertion Palette contains the Step Types list and the Templates list. Use the Templates list to hold copies of steps, variables, and sequences you reuse during the development of sequence files. For example, you can add a step that calls a specific LabVIEW VI you typically use or a sequence that contains common setup steps, cleanup steps, and local variables.

Drag steps from the Steps pane, variables from the Variables pane, and sequences from the Sequences pane and drop them on the Templates list to add steps, variables, or sequences to the Templates list. Use the context menu to rename, copy, paste, delete, import, and export the items in the Templates list. You can drag and drop items to rearrange the list. Select **Insert Folder** from the context menu to add folders to the list. TestStand stores the settings for the Templates list, including any modifications to the list, in the `<TestStand Application Data>\Cfg\Templates.ini` file.

You cannot directly modify templates. Drag a sequence, step, or variable from the Templates list to a sequence file to edit the item. Then, drag the item back to the Templates list and delete the original item from the Templates list.

Inserting a Template Step

Complete the following steps to import a set of template steps to the Insertion Palette, insert a step from the Templates list in a sequence, and add a new step to the Templates list in the `Computer2.seq` sequence file you created in the previous tutorial.

1. Right-click the **Steps** folder in the Templates list of the Insertion Palette and select **Import** from the context menu.
2. Navigate to the `<TestStand Public>\Tutorial` directory, select `Tutorial Templates.ini`, and click **Open**. The sequence editor adds a Tutorial folder under the Steps folder.
3. Expand the **Tutorial** subfolder of the Steps folder to view the following imported template steps:
   - **Retry if Previous Step Fails**—A Message Popup step that prompts you to retry the previous step on failure.
   - **Open Notepad**—A Call Executable step configured to launch the Microsoft Notepad application.
   - **Output Message**—A statement step that logs an output message in which the logged text is the step name.
   - **Waveform Popup**—A step that calls the WaveformGraphPopup function in the LabWindows/CVI user interface library.

4. In the Templates list, select **Retry if Previous Step Fails**, drag the step below the Power On step on the Steps pane, and rename the step `Retry Power On`.
5. Save the changes and select **Execute»Single Pass**.
6. Select the **Power On** test to fail and click **Done**. TestStand executes the Retry Power On step and launches the Step 'Power On' Failed dialog box because the precondition for the step determined that the previous Power On step failed.
7. Click **Retry** in the Step 'Power On' Failed dialog box to execute the Power On step again. The Power On step fails again, and TestStand launches the Step 'Power On' Failed dialog box again.
8. Click **Continue** in the Step 'Power On' Failed dialog box to continue the execution.
9. Review the report. Notice that the Power On step executed twice, and the second call to Retry Power On continued the execution.
10. Close the Execution window. Leave the sequence file open for the next tutorial.

Creating a Template Step

Complete the following steps to update the Retry Power On step so that it automatically selects the **Continue** button if you do not respond to the prompt in the `Computer2.seq` sequence file you created in the previous tutorial.

1. In the Sequence File window, select the **Retry Power On** step on the Steps pane.
2. Click the **Text and Buttons** tab of the Step Settings pane.
3. In the Button Options section, select **Button 2** from the Timeout Button ring control.
4. Enter `20` in the Time To Wait control to instruct the step to wait for 20 seconds before continuing. This technique is useful if an operator is not present to acknowledge a non-critical message during testing.
5. Save the changes and select **Execute»Single Pass**.
6. Select the **Power On** test to fail and click **Done**. TestStand executes the Retry Power On step and launches the Step 'Power On' Failed dialog box. Do not take any action in this dialog box. When the timeout reaches zero, the execution continues as if you clicked **Continue**.
7. In the Sequence File window, drag the Retry Power On step into the Tutorial folder in the Templates list.
8. Rename the new template step `Timeout Retry`. You can use the new template step in subsequent development. The sequence editor automatically saves the templates list when you shut down the sequence editor.
9. Close the `Computer2.seq` sequence file and the Execution window.
10. You no longer need the Tutorial folder in the Templates list, and you can delete the folder if you want. Right-click the **Tutorial** folder in the Templates list and select **Delete** from the context menu to remove the folder.

4

Debugging Sequences

You can debug sequences by tracing, setting breakpoints and conditional breakpoints, stepping through code, and including watch expressions. Refer to the *Using the Watch View Pane* section of Chapter 5, *Using Variables and Properties*, for more information about watch expressions.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Step Mode Execution

Note Before you begin this tutorial, select **Configure»Station Options** to launch the Station Options dialog box, click the **Execution** tab, and confirm that the Enable Tracing and Allow Tracing into Setup/Cleanup options are enabled. Click **OK** to close the Station Options dialog box.

Setting a Breakpoint

Complete the following steps to set a breakpoint in a sequence file.

1. Open `<TestStand Public>\Tutorial\Computer2.seq`, which you created in the previous tutorial.
2. Select **File»Save *<filename>* As** and save the sequence file as `Computer3.seq` in the `<TestStand Public>\Tutorial` directory.
3. Select **Execute»Break on First Step** to suspend an execution on the first step TestStand executes. A checkmark appears to the left of the menu option to indicate that you enabled this option.
4. Expand the **Cleanup** step group on the Steps pane.

   Steps in the Cleanup step group execute regardless of whether the sequence completes successfully or whether a run-time error occurs in the sequence. If a step in the Setup or Main step group causes a run-time error to occur or if the operator terminates the execution, the flow of execution stops and jumps to the Cleanup step group. Steps in the Cleanup group always run even when some of the steps in the Setup group do not run. When a step in the Cleanup group causes a run-time error, execution continues to the next step in the Cleanup group.

5. Insert a Message Popup step in the Cleanup step group and rename the step `Cleanup Message`.
6. On the Text and Buttons tab of the Step Settings pane, enter `"I am now in the Cleanup Step Group."`, including the quotation marks, in the Message Expression expression control.

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

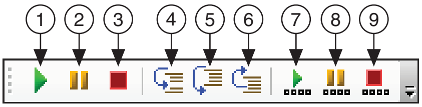
Note For this example, use the default values for all the settings on the Options tab and the Layout tab of the Step Settings pane.

7. Save the changes. Leave the sequence file open for the next tutorial.

Stepping through a Sequence File

When you suspend an execution, you can step through the sequence using the Step Into, Step Over, and Step Out commands on the Debug toolbar, as shown in Figure 4-1.

Figure 4-1. Debug Toolbar Buttons



1	Resume	4	Step Into	7	Resume All
2	Break	5	Step Over	8	Break All
3	Terminate Execution	6	Step Out	9	Terminate All

You can find these and other debugging tools in the Debug menu of the sequence editor.

Complete the following steps to step through an execution of the `Computer3.seq` sequence file you created in the previous tutorial.

1. Select **Execute»Run MainSequence**.

   After the execution starts, the sequence editor immediately suspends at the first step of the sequence because you enabled the Break on First Step option in step 3 of the previous tutorial. The Execution window tab includes an execution pointer icon to indicate the running state of the execution.

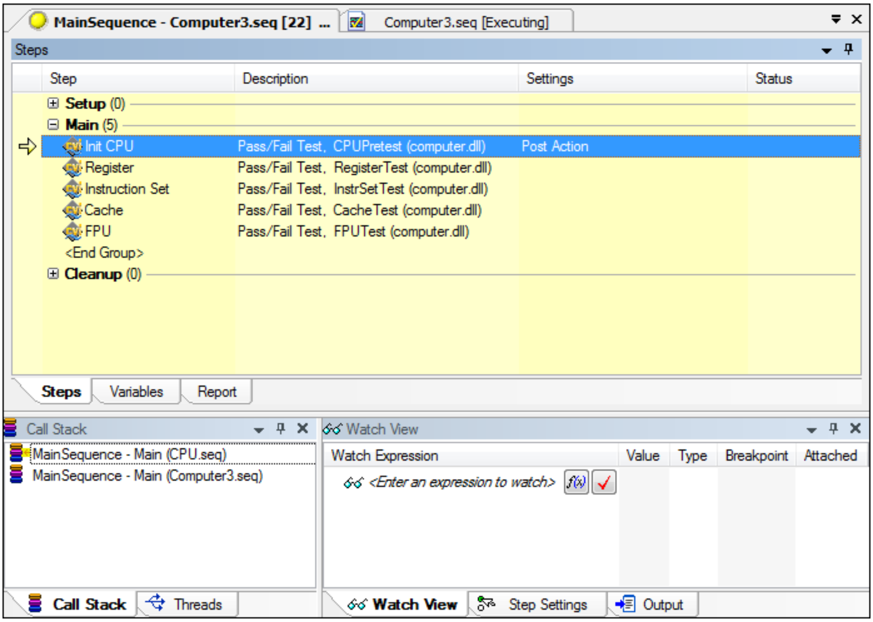
   When the execution suspends, you can step through the sequence using the Step Into, Step Over, and Step Out commands on the Debug toolbar.
2. Click the **Step Over** button to execute the Display Dialog step, which launches the Test Simulator dialog box.
3. Select the **RAM** test to fail and click **Done**. After the Test Simulator dialog box closes, the sequence editor suspends the sequence execution at the end of the Setup step group on `<End Group>`.

4. Insert a breakpoint at the CPU Test step by clicking to the left of the step icon. In the Execution window, a dark red stop sign icon, as shown in the following figure, appears to the left of the CPU Test step to indicate the breakpoint.

Note To specify a conditional breakpoint, right-click the stop sign icon and select **Breakpoint»Breakpoint Settings** to launch the Breakpoint Settings dialog box, in which you can specify an expression that must evaluate to `True` to activate the breakpoint. Click **OK** to close the Breakpoint Settings dialog box. Conditional breakpoints use a bright red stop sign icon in the Sequence File and Execution windows. Disabled breakpoints use a gray stop sign icon. When you globally disable all breakpoints, the breakpoints use a white stop sign icon. Refer to Chapter 5, *Using Variables and Properties*, for more information about expressions.

5. Click the **Resume** button to continue the execution. The sequence editor suspends the execution on the `CPU Test` step.
6. Click the **Step Into** button to step into the `MainSequence` subsequence in `CPU.seq`, and click the **Call Stack** pane. Figure 4-2 shows the Execution window Steps pane and Call Stack pane after you step into the subsequence.

Figure 4-2. Steps Pane and Call Stack Pane while Suspended in Subsequence



MainSequence - Computer3.seq [22] ... | Computer3.seq [Executing]

Steps

Step	Description	Settings	Status
⊞ **Setup** (0)			
⊟ **Main** (5)			
Init CPU	Pass/Fail Test, CPUPretest (computer.dll)	Post Action	
Register	Pass/Fail Test, RegisterTest (computer.dll)		
Instruction Set	Pass/Fail Test, InstrSetTest (computer.dll)		
Cache	Pass/Fail Test, CacheTest (computer.dll)		
FPU	Pass/Fail Test, FPUTest (computer.dll)		
<End Group>			
⊞ **Cleanup** (0)			

Steps | Variables | Report

Call Stack

MainSequence - Main (CPU.seq)
MainSequence - Main (Computer3.seq)

Call Stack | Threads

Watch View

Watch Expression	Value	Type	Breakpoint	Attached
<Enter an expression to watch>				

Watch View | Step Settings | Output

Sequence call steps are similar to subVIs in LabVIEW and function or method calls in C/C++. When a step invokes a subsequence, the sequence that contains the calling step waits for the subsequence to return. The subsequence invocation is nested in the invocation of the calling sequence. The sequence that is currently executing is the most nested sequence. The chain of active sequences that wait for nested subsequences to complete is called the call stack. The first item in the call stack is the most-nested sequence invocation.

The Call Stack pane displays the call stack for the execution thread currently selected on the Threads pane. A yellow pointer icon appears to the right of the most nested sequence invocation while the sequence executes, as shown in Figure 4-2.

7. Select each sequence on the Call Stack pane to view each sequence invocation. Return to the most nested sequence invocation in the call stack, `CPU.seq`.
8. Click the **Step Over** button to step through the `CPU.seq` subsequence one step at a time. Before you reach the end of the `CPU.seq` sequence, click the **Step Out** button. TestStand resumes the execution through the end of the current sequence and suspends the execution at the next step of the calling sequence or breakpoint, whichever comes first.
9. Continue stepping through the `Computer3.seq` sequence by clicking the **Step Over** button until the `Cleanup Message` step you added to the Cleanup step group executes. You must step over each of the 10 loops of the Video Test step.

10. Click **OK** to close the Cleanup Message dialog box and click the **Step Over** button to complete the execution. The Execution window dims when the execution completes. Do not close the Execution window.
11. Click the **Restart** button to rerun the execution. The Execution window must be the active window to restart the execution.
12. After the sequence editor suspends the execution on the first step, click the **Terminate Execution** button. TestStand launches the Cleanup Message dialog box even though you terminated the sequence execution. When an operator or run-time error terminates the execution, TestStand proceeds immediately to the steps in the Cleanup step group. Click **OK** to close the Cleanup Message dialog box. The Execution window dims when the execution completes.
13. Click the **Restart** button to rerun the execution. The Execution window must be the active window to restart the execution.
14. After the sequence editor suspends the execution on the first step, select **Debug»Abort All (no cleanup)**. The execution of the sequence immediately stops, and TestStand does not execute any steps in the Cleanup step group.
15. Save the changes and close all the windows in the sequence editor.

5

Using Variables and Properties

You can create and use variables and properties and monitor the values of the variables and properties.

You can define the following types of variables to share data among steps of a sequence or among several sequences:

- Local variables store data relevant to only the current sequence. Only steps within the sequence that defines the local variable can access these variables.
- Sequence file global variables store data relevant to the entire sequence file. Each sequence and step in the sequence file can directly access these global variables.
- Station global variables persist across different executions. The TestStand Engine maintains the value of station global variables in a file on the computer on which TestStand is running.

In the sequence editor, the Variables pane displays all the variables and properties the selected sequence can access at run time. In the Execution window, the Variables pane displays the sequence context for the sequence invocation currently selected on the Call Stack pane. The sequence context contains all the variables and properties the steps in the selected sequence invocation can access. Use the Variables pane to examine and modify the values of these variables and properties when the sequence is in a suspended state, such as when paused at a breakpoint.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Creating Local Variables

Complete the following steps to create and use local variables. You can apply the concepts you learn in this tutorial to sequence file global and station global variables.

1. Open `<TestStand Public>\Tutorial\Computer2.seq`, which you created in Chapter 3, *Editing Steps in a Sequence*.
2. Select **File»Save *<filename>* As** and save the sequence file as `Computer4.seq` in the `<TestStand Public>\Tutorial` directory.

3. Click the **Variables** pane in the Sequence File window. Expand the **Locals** item to view the local variables currently defined for `MainSequence`.

   Each sequence includes a ResultList local variable, which is an empty array of container properties. TestStand uses this variable to store step results for result processing.

4. On the Variables pane, right-click **Locals**, select **Insert Local»Number** from the context menu to insert a new numeric local variable, and rename the local variable `LoopIndex`.

5. Complete the following steps to insert and configure a For step.

   a. Click the **Steps** pane in the Sequence File window and insert a **Flow Control»For** step below the Retry Power On step. Notice that TestStand also adds an End step below the For step.

   b. Drag the **End** step below the RAM step. TestStand automatically indents the Retry Power On, CPU Test, ROM, and RAM steps between the For and End steps.

   c. Click the **For** step and click the **For Loop** tab of the Step Settings pane.

   d. Enter `5` in the **Number of Loops** control.

   e. For the Loop Variable control, click the **Expression Browser** button, as shown in the following figure, to launch the Expression Browser dialog box, in which you can interactively build an expression by selecting from lists of variables, properties, operators, functions, and the TestStand API.

   TestStand supports all applicable expression operators and syntax you can use in C, C++, Java, and Visual Basic .NET. You can also call the TestStand API directly from within expressions.

   All TestStand controls that accept expressions provide context-sensitive editing features, such as drop-down lists, syntax checking, and expression coloring to help you create expressions. At any point while editing an expression, you can press <Ctrl-Space> to show a drop-down list of valid expression elements.

   f. Expand the **Locals** item on the Variables/Properties tab of the Expression Browser dialog box. Each item in the top section of the Variables/Properties tab is a property or variable of TestStand.

   g. Select the **LoopIndex** variable under the Locals property and click the **Insert** button. The Expression Browser enters `Locals.LoopIndex` in the Expression control.

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

h. Click the **Check Expression for Errors** button, as shown in the following figure, to verify that the expression contains valid syntax.

i. In the Expression Browser dialog box, click **OK** to return to the For Loop tab of the Step Settings pane. The Loop Variable control now contains the `Locals.LoopIndex` expression. Notice that the Custom Loop section shows the expressions TestStand uses when executing the For step when using a fixed number of loops.

6. Select **Execute»Break on First Step** to remove the checkmark that appears to the left of the menu item and disable this option, which you enabled in the *Step Mode Execution* section of Chapter 4, *Debugging Sequences*, of this manual.
7. Save the changes and select **Execute»Single Pass**.
8. Click **Done** in the Test Simulator dialog box.
9. After the sequence executes, review the test report, which shows that TestStand executed the steps within the loop (Retry Power On, CPU Test, ROM, and RAM) five times.
10. Close the Execution window. Leave the sequence file open for the next tutorial.

Using the Execution Window Variables Pane

Before executing the steps in a sequence, TestStand creates a run-time copy of the sequence to maintain separate local variable and step property values for each sequence invocation. When an execution completes, TestStand discards the run-time sequence copy.

For each active sequence, TestStand maintains a sequence context that contains references to the run-time copy of the sequence so you can access all the objects, variables, and properties that relate to the execution of the sequence.

The contents of the sequence context vary depending on the currently executing sequence and step, the location of the active sequence in the call stack, and the identity of the execution in which the active sequence resides. Depending on the current state of execution, sequence context subproperties might not exist. When a property exists, the contents of the property can vary.

The Variables pane displays the sequence context for the sequence invocation currently selected on the Call Stack pane. The sequence context contains all the variables and properties the steps in the selected sequence invocation can access. Use the Variables pane to examine and modify the values of these variables and properties when the sequence is in a suspended state, such as when paused at a breakpoint.

Table 5-1 lists the top-level properties in the sequence context.

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

Sequence Context Subproperty	Description
`Locals`	Contains local variables in the current sequence. Only the current sequence can access these variables.
`Parameters`	Contains parameter variables in the current sequence. Only the current sequence or calling sequences can access these variables.
`FileGlobals`	Contains file global variables in the current sequence file. All the sequences in the current sequence file can access these variables.
`StationGlobals`	Contains station global variables on the computer. Any sequence on the current computer can access these variables. Station global variables are stored on disk and the values persist even after you close TestStand.
`ThisContext`	Contains a reference to the current sequence context. You typically use this property to pass the entire sequence context as an argument to a subsequence or a step code module.
`RunState`	Contains properties that describe the current state of execution.
`Step`	Contains the properties in the currently executing step. The Step property exists only while a step executes. The property does not exist when the execution is between steps, such as at a breakpoint.

Complete the following steps to use the Variables pane of the Execution window to examine the value of the LoopIndex variable while TestStand executes the `Computer4.seq` sequence file you created in the previous tutorial.

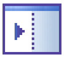
1. Insert a breakpoint at the End step associated with the For loop you created in the previous tutorial.
2. Select **Execute»Single Pass**.
3. Click **Done** in the Test Simulator dialog box. The execution suspends on the End step.
4. Click the tab for the Variables pane of the Execution window and expand the **Locals** section.
5. Select the **LoopIndex** property. The numeric value of LoopIndex is `0`.
6. Click the tab for the Steps pane of the Execution window and click the **Resume** button on the Debug toolbar. The execution resumes and suspends at the End step again.
7. Click the tab for the Variables pane again. The value of `Locals.LoopIndex` is now `1`. Leave the execution in the suspended state for the next tutorial.

Docking the Variables Pane

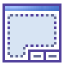
The sequence editor contains tabbed windows and panes you can float, dock, resize, and hide. Some panes, such as the Variables pane, must stay attached to the associated Sequence File or Execution window.

Complete the following steps to display the Steps and Variables panes at the same time.

1. In the Execution window, click the tab for the Variables pane and drag it to the right using the tab, not the titlebar. As you drag the pane from the current location, the sequence editor detaches the pane and displays docking guides. The docking guides show you where you can place the pane. As you move the mouse over a docking guide, the sequence editor highlights where the pane will relocate when dropped.
2. Drop the pane on the right-most docking guide, as shown in the following figure, so the pane appears to the right of the Steps panes.



3. Click the tab for the Steps pane and click the **Resume** button on the Debug toolbar. The execution resumes and suspends at the End step again. The value of `Locals.LoopIndex` on the Variables pane is now `2`.
4. Drag the titlebar of the Variables pane and drop it on the center docking guide, as shown in the following figure, to return the Variables pane to the previous location.



   Notice that the tab for the Variables pane now appears to the left of the tab for the Steps pane instead of to the right.
5. Click the tab for the Variables pane, drag it to the right, and drop it on the Steps pane tab to adjust the order of the tabs. Leave the execution in the suspended state for the next tutorial.

Note Select **View»Reset UI Configuration** at any time to restore the panes to the original state.

Using the Watch View Pane

The Watch View pane of the Execution window displays the values of watch expressions you enter. The values in the Watch View pane update when execution suspends at a breakpoint. When you enable tracing, the sequence editor also updates the values after each step executes. The Watch View pane highlights viewed and changed watch expression values in red text.

Enter watch expressions to monitor values of variables and properties as you trace or step through a sequence. You can drag individual variables or properties from the Variables pane to the Watch View pane. When you specify a container property, array property, or a PropertyObject reference as a watch expression, you can use the Watch View pane to expand the subproperties to view the values.

Complete the following steps to create a watch expression that uses the `LoopIndex` property.

1. Select the **LoopIndex** property on the Variables pane of the Execution window and drag the property to the Watch View pane. The value of the `LoopIndex` watch expression is `2`.
2. Edit the `LoopIndex` watch expression directly in the Watch Expression column on the Watch View pane to change the expression to the following:

   ```
   Str (Locals.LoopIndex * 20) + "%"
   ```

3. Click the tab for the Steps pane of the Execution window and click the **Resume** button on the Debug toolbar. The execution resumes and suspends at the End step again. The value of the watch expression changes from `40%` to `60%`.

   Note The watch expression pane might display an **`Error in argument`** message before displaying `60%` because the watch expression is not valid during a subsequence execution.

4. Click the breakpoint icon to the left of the End step in the Execution window to remove the breakpoint.
5. Resume and complete the execution.
6. Close the Execution window.
7. Select **Debug»Breakpoints/Watches** to launch the Edit Breakpoints/Watch Expressions dialog box, in which you can review and edit the breakpoints and watch expressions associated with the current workspace. TestStand saves and reloads breakpoints and watch expressions using an options file associated with the current workspace or a default options file when no workspace is loaded.
8. Review the contents of the Breakpoints and Watch Expressions tabs in the Edit Breakpoints/Watch Expressions dialog box, delete the breakpoint and expression you created for this tutorial, and click **Done** to close the dialog box.
9. Close the sequence file.

6 Using Callback Sequences

Callbacks are sequences TestStand calls under specific circumstances. You can create new callbacks or you can override existing callbacks to customize the operation of the test station.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

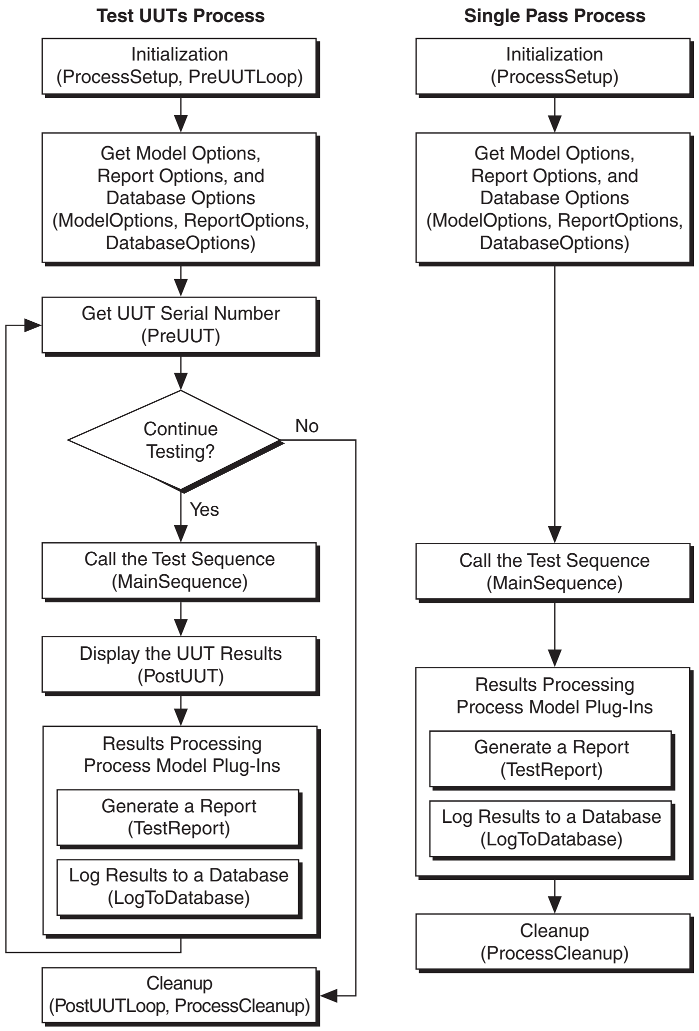
Process Model Callbacks

The process models contain sequences that define the operations TestStand performs before and after testing a UUT. You can use an Execution entry point, such as Test UUTs or Single Pass, to invoke sequences within a process model to execute a sequence in a sequence file you create. The process models also contain Model callbacks, which are sequences you can override to customize the behavior of a process model without editing the process model directly.

For example, the process models define a PreUUT callback sequence that prompts the user for a serial number before the test sequence executes. In most cases, this default behavior is sufficient. You can override the default PreUUT callback in a sequence file you create. When you override the default PreUUT callback and you use the Test UUTs or Single Pass Execution entry point to execute a sequence file, the process model calls the custom PreUUT callback in the sequence file instead of the default PreUUT callback in the process model.

Figure 6-1 shows the actions the default Sequential process model performs using callback sequences within the Test UUTs and Single Pass Execution entry points.

Figure 6-1. Model Actions Performed in Sequential Model Callbacks


Test UUTs Process
Initialization (ProcessSetup, PreUUTLoop)
Get Model Options, Report Options, and Database Options (ModelOptions, ReportOptions, DatabaseOptions)
Get UUT Serial Number (PreUUT)
Continue Testing?
No
Yes
Call the Test Sequence (MainSequence)
Display the UUT Results (PostUUT)
Results Processing Process Model Plug-Ins
Generate a Report (TestReport)
Log Results to a Database (LogToDatabase)
Cleanup (PostUUTLoop, ProcessCleanup)
Single Pass Process
Initialization (ProcessSetup)
Get Model Options, Report Options, and Database Options (ModelOptions, ReportOptions, DatabaseOptions)
Call the Test Sequence (MainSequence)
Results Processing Process Model Plug-Ins
Generate a Report (TestReport)
Log Results to a Database (LogToDatabase)
Cleanup (ProcessCleanup)

You can modify the process models or replace the models entirely to alter the behavior of the process models for all sequences.

Note To modify the installed process models or to create a new process model, copy all the process model files from the `<TestStand>\Components\Models\TestStandModels` directory to the `<TestStand Public>\Components\Models\TestStandModels` directory and make changes to the copy. Use the following guidelines when you copy installed files to modify:

- You must rename the files after you modify them if you want to create a separate custom component. You must register any new or renamed ActiveX components.
- You do not have to rename the files after you modify them if you only want to modify the behavior of an existing component.
- If you do not rename the files and you use the files in a future version of TestStand, changes National Instruments makes to the component might not be compatible with the modified version of the component.
- Storing new and customized files in the `<TestStand Public>` directory ensures that installations of the same version of TestStand do not overwrite the customizations and ensures that uninstalling TestStand does not remove the files you customize.

Viewing Process Model Callbacks

Complete the following steps to examine the Model callbacks in the Sequential process model, which is the default process model TestStand uses to execute sequences.

1. Open `<TestStand>\Components\Models\TestStandModels\SequentialModel.seq`.
2. On the Sequences pane, select **Test UUTs**, which is the Execution entry point TestStand executes when you select **Execute»Test UUTs**. The Main step group of the Test UUTs sequence calls several callback sequences, including the following:
   - PreUUTLoop callback
   - PreUUT callback
   - MainSequence callback
   - PostUUT callback
   - PostUUTLoop callback
3. Double-click the **PreUUT Callback** step, not the PreUUTLoop Callback step, to view the PreUUT callback sequence, which calls the DoPreUUT sequence.
4. On the Sequences pane, select the **DoPreUUT** sequence. If the DoPreUUT Properties dialog box launches, click **OK** to close it. The DoPreUUT sequence includes an IdentifyUUT step and a Set Serial Number step.

Note If you double-click the Call DoPreUUT step on the Steps pane of the PreUUT Callback sequence to try to open the DoPreUUT sequence, TestStand returns an error that it cannot open the sequence because TestStand evaluates an expression at run time to determine the sequence file path.

5. Right-click the **IdentifyUUT** step and select **Run Selected Steps** from the context menu to launch the UUT Information dialog box, which is similar to the dialog box that launches when you execute a sequence using the Test UUTs Execution entry point. You can override the PreUUT callback with a custom callback to change the way TestStand obtains a UUT serial number, such as reading the serial number from a bar code instead.
6. Click **OK** in the UUT Information dialog box.
7. Close the Execution window.
8. On the Sequences pane, select the **Test UUTs** sequence.
9. Double-click the **PreUUTLoop Callback** step to open the PreUUTLoop callback sequence, which is empty because it is a placeholder. If you want to add steps that execute before the UUT loop, you can complete the steps in the *Overriding a Process Model Callback* section of this chapter to override this callback.
10. Close `SequentialModel.seq` and decline any prompts to save the changes.

Overriding a Process Model Callback

You can customize the functionality of a process model without making changes to the model itself. When you override a callback, the process model invokes the callback you create in a sequence file instead of the default callback in the process model.

Complete the following steps to override the default PreUUTLoop callback sequence in the Sequential model by creating a PreUUTLoop callback sequence in a client sequence file. Use this technique when you want to perform a task only once before operating on multiple UUTs, such as initializing hardware.

1. Open `<TestStand Public>\Tutorial\Computer4.seq`, which you created in Chapter 5, *Using Variables and Properties*.
2. Select **File»Save *<filename>* As** and save the sequence file as `Computer5.seq` in the `<TestStand Public>\Tutorial` directory.
3. Select **Edit»Sequence File Callbacks** to launch the Sequence File Callbacks dialog box.
4. Select the **PreUUTLoop** callback and click **Add**. The value in the Present column changes from `no` to `yes`. Click **OK** to close the Sequence File Callbacks dialog box. The sequence editor creates a new empty callback sequence in the sequence file. Now, when you start an execution using an Execution entry point, TestStand calls the callback in the sequence file instead of the callback sequence in the Sequential process model file.

   The PreUUTLoop callback sequence becomes the selected sequence on the Sequences pane and opens in the Sequence File window.

5. Insert a Message Popup step in the Main step group of the PreUUTLoop callback sequence and rename the new step `Pre UUT Loop Callback Message`.
6. Enter the literal string `"Now in the Pre UUT Loop Callback"` in the Message Expression control.
7. Save the changes and select **Execute»Test UUTs**. TestStand launches the Pre UUT Loop Callback Message dialog box.
8. Click **OK** to close the Pre UUT Loop Callback Message dialog box. TestStand launches the UUT Information dialog box from the PreUUT callback sequence in the Sequential model. Enter a serial number and click **OK**.
9. Run through several iterations of the sequence. TestStand launched the Pre UUT Loop Callback Message dialog box only once at the very beginning of the execution because the PreUUTLoop callback executes before the loop, and the PreUUT callback executes within the loop.
10. Click **Stop** in the UUT Information dialog box.
11. Close all the windows in the sequence editor.

The Parallel and Batch process models include similar callbacks that you can override.

7

Adding Users and Setting Privileges

Use the TestStand User Manager to maintain the list of users, user names, user passwords, user privileges, groups, group privileges, and members of groups. TestStand can limit the functionality of the TestStand Sequence Editor and User Interfaces depending on the privilege settings you specify in the user manager for the current user and the groups to which the user belongs.

The user manager helps you implement policies and procedures that concern the use of test stations. The user manager is not a security system, and it does not inhibit or control the operating system or third-party applications. Use the system-level security features the operating system provides to secure test station computers against unauthorized use.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Adding a New User

Note This tutorial assumes you are currently logged in to TestStand as the `administrator` user. If you are not logged in as administrator, select **File»Login** and select **administrator** from the User Name ring control. Leave the Password field empty and click **OK**.

Complete the following steps to add a new user.

1. Click the **User Manager** button on the Environment toolbar to open the User Manager window, which shows all the users and groups configured on the test station.
2. Right-click the **Users** item, select **Insert User** from the context menu, and enter a name to create a new user.
3. Right-click the new user you just added, select **Properties** from the context menu to launch the <*User*> Properties dialog box, and complete the following steps.
    a. Confirm that the **User Name** control displays the new name you entered.
    b. Enter the full name in the **Full Name** control.

   c. Enter a password in the **Password** and **Confirm Password** controls.
   d. Select **Operator** in the Group Privileges control.
   e. Click **OK** to close the User Properties dialog box.
4. Save the changes to the user manager.
5. Select **File»Login** to launch the Login dialog box. The User Name ring control now includes the new user you just added.
6. Select the user you just created, enter the appropriate password, and click **OK**.
7. Open `<TestStand Public>\Tutorial\Computer.seq`.
8. Select the **Execute** menu and notice that the Single Pass and Run MainSequence options of the Execute menu are disabled because the user you just created does not have the privileges to execute them.
9. Right-click the **Steps** pane to insert a new step. The Insert Step menu command is also disabled because the user privileges have changed.
10. Select **File»Login** and select **administrator** from the User Name ring control. Leave the Password field empty and click **OK**.

Creating a New Group

You can use the user manager to modify the default groups and to create new groups that define a combination of appropriate privileges.

The default Operator, Technician, Developer, and Administrator groups define a set of privilege settings for the new user to inherit. By default, the Operator group grants a user the privilege to execute, terminate, and abort sequences but does not grant the privilege to create or debug sequences.

Complete the following steps to create a new group.

1. Open the User Manager window.
2. Expand the **Groups** item to show the four default groups.
3. Right-click the **Operator** group and select **Copy** from the context menu.
4. Right-click the **Groups** item and select **Paste** from the context menu.
5. Rename the new group `Senior Operator`. The new group is identical to the Operator group except for the name.

Modifying Privileges

The TestStand User Manager stores user and group privileges as Boolean properties and organizes the privileges in the following categories:

- **Operate**—Privileges for executing sequences and terminating and aborting executions.
- **Debug**—Privileges for controlling execution flow, executing manual and interactive executions, and editing station global variables and run-time variables.

- **Develop**—Privileges for editing and saving sequence files, editing and saving workspace files, and using source code control.
- **Configure**—Privileges for editing process model files and configuring station options, users, adapters, application settings, and report, database logging, and model options.
- **Custom**—Custom privileges you define. Customize the `NI_UserCustomPrivileges` data type to add new privileges.

You can grant all privileges in a specific category for each user or group in the user manager, and you can grant specific privileges for each user or group. In addition, when you add a user as a member of a group, TestStand grants the user all the privileges of the group. TestStand grants a privilege to a user or group when the property value for the privilege is `True` or when the value of the `GrantAll` property in any enclosing parent privilege category is `True`. For example, a user has the privilege to terminate an execution when one of the following properties is `True`:

- `<`*`User`*`>.Privileges.Operate.Terminate`
- `<`*`User`*`>.Privileges.Operate.GrantAll`
- `<`*`User`*`>.Privileges.GrantAll`
- `<`*`Group`*`>.Privileges.Operate.Terminate`
- `<`*`Group`*`>.Privileges.Operate.GrantAll`
- `<`*`Group`*`>.Privileges.GrantAll`

Note TestStand stores the privilege categories as subproperties of the `Privileges` property. The `Privileges` property also includes a Boolean `GrantAll` subproperty. The property `Privileges.GrantAll` applies to all privilege categories. When you set the `GrantAll` property to `True`, the user or group has all privileges. You must set the `GrantAll` property to `False` to honor privilege settings within each privilege category.

Complete the following steps to modify the default privileges for the group you created in the previous section of this tutorial.

1. Expand the **Senior Operator** item and expand the **Privileges** property.
2. Expand the **Debug** item, which is a property that contains Boolean subproperties. Use the **Value** column ring control to change the `SinglePass` property under Debug to `True`.
3. Complete the following steps to add the user you created in the *Adding a New User* section of this tutorial to the Senior Operator group.
   a. Right-click the user you previously created under Users and select **Properties** from the context menu to launch the <*User*> Properties dialog box.
   b. Disable **Operator** in the Group Privileges control and enable **Senior Operator** instead.
   c. Click **OK** to close the User Properties dialog box.
4. Save the changes to the user manager.
5. Select **File»Login** to launch the Login dialog box. Select the user you previously created, enter the appropriate password, and click **OK**.

6. Select the `Computer.seq` window.
7. Select the **Execute** menu. Notice that the Single Pass option is now available, but the Run MainSequence option is disabled because the user you created does not have the privilege to execute sequences without a Model entry point.
8. Close all the windows in the sequence editor.
9. Select **File»Login** and select **administrator** from the User Name ring control. Leave the Password field empty and click **OK**.

Refer to the `Creating & Deleting Users Using API.seq` example located in the `<TestStand Public>\Examples\TestStand API\Creating & Deleting Users Using APIs` directory for information about how to use the TestStand API to add and remove users programmatically.

8 Interactive Executions

When you run steps in interactive mode, you can execute specific steps in a sequence. Use the Interactive Executions section on the Execution tab of the Station Options dialog box to control if an interactive execution records results, runs steps in the Setup and Cleanup step groups, and evaluates preconditions. The options in the Interactive Executions section also determine how TestStand handles step failures, errors, and branching during interactive executions.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Running Selected Steps as a Separate Execution

Complete the following steps to run selected steps from a Sequence File window as a separate execution.

1. Open `<TestStand Public>\Tutorial\Computer.seq`.
2. Insert a breakpoint at the Power On step.
3. Press <Ctrl> and select the **Power On**, **ROM**, and **ROM Diagnostics** steps.
4. Select **Execute»Run Selected Steps** to start a root interactive execution.

   By default, when you run selected steps from a Sequence File window, TestStand also executes the Setup and Cleanup step groups. You enable or disable the Run Setup and Cleanup option on the Execution tab of the Station Options dialog box to control whether TestStand runs the Setup and Cleanup step groups as part of the root interactive execution.
5. Click **Done** in the Test Simulator dialog box. The execution stops at the Power On step breakpoint. The execution pointer for the interactive execution is a narrow yellow arrow.
6. Step over twice to step through the execution until you reach the ROM Diagnostics step. Only the steps you selected execute. TestStand dims the other steps.
7. Resume and complete the execution.
8. Close the Execution window.

9. Ensure that the Power On, ROM, and ROM Diagnostics steps are selected. Repeat steps 4 through 8, but select **Execute»Run Selected Steps Using»Single Pass** in step 4.

   TestStand executes the steps you selected using the Single Pass Execution entry point, which generates a UUT report.
10. Close the Execution window.

Running Selected Steps During an Execution

Complete the following steps to interactively execute selected steps in a sequence while suspended at a breakpoint during an execution.

1. Ensure that the Power On, ROM, and ROM Diagnostics steps are selected and select **Execute»Single Pass**.
2. Select the **ROM** test to fail and click **Done**. The execution stops at the breakpoint on the Power On step.
3. Step through the execution until you reach the RAM Diagnostics step. Notice that the ROM step failed.
4. Place a breakpoint at the ROM step in the Execution window and select the **ROM** and **ROM Diagnostics** steps.
5. Right-click the **ROM Diagnostics** step and select **Loop on Selected Steps** from the context menu to launch the Loop on Selected Steps dialog box. Enter `100` in the Loop Count control and click **OK**.

   TestStand starts an interactive execution of 100 loops for the steps you selected and enters a suspended state at the breakpoint for the ROM step. The execution pointer for the normal execution remains on the RAM Diagnostics step, and an execution pointer for the new interactive execution points to the ROM step.
6. Step through the interactive execution. The interactive execution toggles only between the ROM step and the ROM Diagnostics step. The ROM step continues to fail.
7. Click the **Terminate Interactive Execution** button on the Debug toolbar. TestStand returns the execution to a suspended state on the RAM Diagnostics step.
8. Complete the following steps to force the execution to continue from a step other than the currently suspended step.
   a. Select the **ROM** step so it is the only highlighted step.
   b. Right-click the **ROM** step and select **Set Next Step to Cursor** from the context menu. The execution pointer moves from the RAM Diagnostics step to the ROM step. The execution continues from the ROM step when you resume or step through the sequence.
   c. Step over once. The ROM step executes instead of the RAM Diagnostics step.
9. Resume and complete the execution. The report contains entries for each step you executed interactively.
10. Remove the breakpoint from the Power On step in the Sequence File window.
11. Close all the windows in the sequence editor and do not save any changes.

9 Calling Sequences Dynamically

You can add to a sequence a step that dynamically runs one of two sequences, and you can pass parameters to the sequence you call.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Dynamically Specifying a Sequence to Run

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

1. Open `<TestStand Public>\Tutorial\Computer.seq`.
2. Select **File»Save *<filename>* As** and save the sequence as `Computer6.seq` in the `<TestStand Public>\Tutorial` directory.
3. Click the **Variables** pane in the Sequence File window, right-click the **Locals** item, select **Insert Local»String** from the context menu, and name the local variable `CPUType`.
4. Click the **Steps** pane in the Sequence File window to show the steps in the `MainSequence`.
5. Insert a Message Popup step below the Power On step, rename the new step `Select CPU Type`, and complete the following steps to configure the Select CPU Type step.
   a. On the Text and Buttons tab of the Step Settings pane, enter the following values in the corresponding controls:
      - **Message Expression**—`"Please select the CPU type for the UUT."`
      - **Button 1**—`"INTEL CPU"`
      - **Button 2**—`"AMD CPU"`
      - **Cancel Button**—`None`

   b. On the Layout tab of the Step Settings pane, enable the **Make Modal** option in the Dialog Options section. Enabling this option prevents the sequence editor from hiding the Select CPU dialog box and prevents you from interacting with the sequence editor until you close the Select CPU dialog box.

   c. On the Properties tab of the Step Settings pane, click **Expressions** to show the Expressions panel.
   d. In the Post-Expression control, enter the following expression to assign the string value `"AMD"` or `"INTEL"` to the local variable depending on the button users click:

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

      Click the **Check Expression for Errors** button to verify that the expression contains valid syntax.
6. On the Steps pane, insert a Message Popup step below the Select CPU Type step, rename the new step `Specify Number of CPUs`, and complete the following steps to configure the Specify Number of CPUs step.
   a. On the Text and Buttons tab of the Step Settings pane, enter the following values in the corresponding controls:
      - **Message Expression**—`"Please select the number of CPUs installed for the UUT."`
      - **Button 1**—`"1"`
      - **Button 2**—`"2"`
      - **Button 3**—`"3"`
      - **Button 4**—`"4"`
      - **Cancel Button**—`None`
   b. On the Layout tab of the Step Settings pane, enable the **Make Modal** option in the Dialog Options section.
7. Insert a Sequence Call step below the Specify Number of CPUs step, rename the step `CPU Test`, and complete the following steps to configure the CPU Test step.
   a. On the Module tab of the Step Settings pane, enable the **Specify By Expression** option.
   b. Enter `Locals.CPUType + "Processor.seq"` in the File Path or Reference control and `"MainSequence"` in the Sequence control.
   c. Click the **Load Prototype** button to launch the Load Sequence Prototype dialog box, in which you can select the prototype for the Sequence Call step.
   d. Click the **Browse** button in the Load Sequence Prototype dialog box and select `<TestStand Public>\Tutorial\AMDProcessor.seq`.
   e. Click **OK** to close the Load Sequence Prototype dialog box. TestStand populates the Parameters Table on the Module tab with the parameter list for the sequence.
   f. Click in the Value column of the **CPUsInstalled** parameter and enter the following expression:

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

      Click the **Check Expression for Errors** button to verify that the expression contains valid syntax.
8. Save the changes. Leave the sequence file open for the next tutorial.

Running a Sequence Dynamically

Complete the following steps to run a sequence dynamically.

1. Place a breakpoint at the CPU Test step.
2. Select **Execute»Single Pass**.
3. Click **Done** in the Test Simulator dialog box.
4. Click the **INTEL CPU** button in the Select CPU Type dialog box and click the **2** button in the Specify Number of CPUs dialog box.
5. When execution suspends at the breakpoint on the CPU Test step, click the **Step Into** button on the Debug toolbar to step into the `MainSequence` sequence in `INTELProcessor.seq`.
6. Click the **Call Stack** pane. The call stack list shows `INTELProcessor.seq` at the top of the sequence call stack.
7. Click the tab for the Variables pane of the Execution window and expand the **Parameters** item. The value of the **CPUsInstalled** parameter equals the value on the button you clicked in the Specify Number of CPUs dialog box. The `MainSequence` sequence in the `INTELProcessor.seq` sequence file also requires a **ModelName** parameter. The Sequence Call step you created did not specify the **ModelName** parameter, so the TestStand Engine initializes the parameter to the default name.
8. Click the tab for the Steps pane of the Execution window and click the **Resume** button on the Debug toolbar. When the execution completes, review the report but do not close the Execution window.
9. Click the **Restart** button on the Debug toolbar to restart the execution and click **Done** in the Test Simulator dialog box. The Execution window must be the active window to restart the execution.
10. Click the **AMD CPU** button in the Select CPU Type dialog box and click the **3** button in the Specify Number of CPUs dialog box.
11. When the execution suspends at the breakpoint on the CPU Test step, step into the `MainSequence` sequence in `AMDProcessor.seq`. The Call Stack pane lists `AMDProcessor.seq` at the top of the call stack.
12. Resume and complete the execution and review the report.
13. Close all the windows in the sequence editor.

10 Customizing Reports

You can customize report generation within TestStand in a variety of ways. You can also create a custom results format by creating a custom result processing plug-in.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Configuring Test Report Options

Complete the following steps to configure the test report options.

1. Open `<TestStand Public>\Tutorial\Computer.seq`.
2. Select **Configure»Result Processing** to launch the Result Processing dialog box.
3. Click the icon in the Options column for the built-in Report model plug-in to launch the Report Options dialog box and complete the following steps.
   a. On the Contents tab, select **ATML 6.01 Standards Report Document** from the Report Format ring control.
   b. Enable the **Include Step Results** option and confirm the following step result settings:
      - Enable the **Include Test Limits** option.
      - Enable the **Include Measurements** option.
      - Select **Insert Graph** from the Include Arrays ring control.
   c. Click the **Edit Format** button located to the right of the Default Numeric Format control to launch the Numeric Format dialog box, in which you specify the format TestStand uses to display the value of a numeric variable or property. By default, TestStand configures the numeric format to report numbers with 13 digits of precision.
   d. Change **Maximum Number of Significant Digits** to `2` and click **OK** to close the Numeric Format dialog box.
4. Click the **Report File Pathname** tab, which you use to specify the report file pathname. You can specify a fixed pathname to use for all report files or you can specify options the report generator uses to generate report file pathnames. Use the default values for the options on this tab and click **OK** to close the Report Options dialog box.
5. Click **OK** to close the Result Processing dialog box.
6. Select **Execute»Test UUTs**. Run through several iterations of the sequence, selecting components other than the Video and CPU tests to fail.

7. Click **Stop** in the UUT Information dialog box to stop sequence execution. The test report contains failure chain information for UUTs that fail. The failure chain shows the step failure that caused the UUT to fail and shows the Sequence Call steps through which the execution reached the failing step. Each step name in the failure chain links to the section of the report that shows the result for the step.
8. Close the Execution window.
9. Select **Configure»Result Processing** to launch the Result Processing dialog box.
10. Click the icon in the Options column for the built-in Report model plug-in to launch the Report Options dialog box and complete the following steps.
    a. Click the **Contents** tab and select **ASCII Text File** from the Report Format ring control.
    b. Select **Exclude Passed/Done/Skipped** from the Result Filtering Expression ring control for TestStand to apply a filtering expression to determine what steps appear in the report.

       In this example, you configure TestStand to record only the results of the steps that do not pass or steps that complete without any status. TestStand also changes the value of the Include Arrays ring control to `Insert Table` because ASCII reports do not support graphs.
11. Click **OK** to close the Report Options dialog box, and click **OK** to close the Result Processing dialog box.
12. Repeat steps 6 through 8 and examine the text version of the test report. Close the Execution window when you finish reviewing the report.

Using External Report Viewers

You can view the test report in external applications more suited for showing and editing text, such as Microsoft Word or Microsoft Excel.

Complete the following steps to use an external report viewer to view the test report.

1. Select **Configure»External Viewers** to launch the Configure External Viewers dialog box.
2. Click the **Add** button.
3. Click the **Browse** button located to the right of the **Viewer** control and navigate to the application you want to use to view the report, such as Word, and click **OK**.

Note When you select the application you want to use and click **OK**, TestStand might launch the File Not Found dialog box because TestStand cannot locate the file within the default TestStand search directories, which TestStand uses to resolve relative paths to code modules and other files or directories. Select the **Use an absolute path for the file you selected** option to ensure that TestStand can access the application you want to use to view the report, and click **OK** to close the File Not Found dialog box.

4. Select **txt** from the Format pull-down menu.
5. Enable the **Automatically Launch Viewer** option.
6. Click **OK** to close the Configure External Viewers dialog box.
7. Select **Execute»Test UUTs**. Run through several iterations of the sequence, selecting components other than the Video and CPU tests to fail.
8. Click **Stop** in the UUT Information dialog box to stop sequence execution. TestStand generates the text report and launches the external viewer application you configured in step 3 to show the test report.
9. Examine the test report and close the external report viewing application.
10. Close the Execution window.
11. Complete the following steps to change the report settings back to the default settings.
    a. Select **Configure»Result Processing** to launch the Result Processing dialog box.
    b. Enable the **Show More Options** option.
    c. Click the **Reset to Defaults** button.
    d. Click **OK** in the Reset Configuration prompt.
    e. Click **OK** to close the Result Processing dialog box.
12. Complete the following steps to change the external viewer settings back to the default settings.
    a. Launch the Configure External Viewers dialog box.
    b. Select the external viewer application you configured in step 3 and click the **Delete** button.
    c. Click **OK** to close the Configure External Viewers dialog box.
13. Close all the windows in the sequence editor.

Adding Additional Results to Reports

Use the Additional Results panel to add and configure additional results, which are values TestStand adds to the result list of a step when the step executes. An additional result can be a module parameter or a custom additional result in which you specify the name and value of the result. You can configure TestStand to automatically include additional results when generating a report or when logging results to a database. The default TestStand report generator style sheets do not display additional results for skipped steps.

Complete the following steps to create a step that calls a DLL code module to return a numeric array and add the numeric array and other values to the report.

1. Select **File»New»Sequence File** to open a new sequence file.
2. Save the sequence file as `CustomReport.seq` in the `<TestStand Public>\Tutorial` directory.

3. Complete the following steps to create a local variable.
   a. Click the **Variables** pane in the Sequence File window, right-click the **Locals** item and select **Insert Local»Array of»Number** from the context menu to launch the Array Bounds dialog box, in which you can modify the array bounds.
   b. Enter `49` in the Upper Bounds control and click **OK**.
   c. Rename the local variable `NumArray`.
4. Click the **C/C++ DLL** adapter icon, as shown in the following figure and located at the top of the Insertion Palette.
5. Insert an Action step in the Main step group and complete the following steps to configure the Action step.
   a. On the Module tab of the Step Settings pane, click the **Browse** button located to the right of the Module control, navigate to `<TestStand Public>\Tutorial\NumericArray.dll`, and click **Open**.
   b. Select **GetNumericArray** from the Function ring control.
   c. In the Value Expression column of the **measurements** parameter in the Parameters Table, enter `Locals.NumArray` to copy the value from the numeric array output parameter to the `Locals.NumArray` variable when TestStand returns from calling the `GetNumericArray` function.
   d. In the Value column of the `Dim 1 Size` property in the Parameter Details Table located to the right of the Parameters Table, enter `-1` to specify that TestStand passes all elements of the `Locals.NumArray` property to the code module.
6. On the Properties tab of the Step Settings pane, complete the following steps to add the numeric array and other values to the report.
   a. Click **Additional Results** to show the Additional Results panel.
   b. Place a checkmark in the **measurements [Out]** parameter checkbox to log the output value of the parameter.
   c. Click the **Add Custom Result** button to add a new empty row to the list of the results.
   d. Enter `"Time"`, including the quotation marks, in the Name column and enter `Time()` in the Value to Log column. The `Time` TestStand expression function returns the current time.
   e. Click the **Add Custom Result** button again.
   f. Enter `"Date"`, including the quotation marks, in the Name column and enter `Date()` in the Value to Log column. The `Date` TestStand expression function returns the current date.
7. Save the changes you made and select **Execute»Single Pass**. Review the report that includes the array data, the time, and the date.
8. Close all the windows in the sequence editor.

Adding to Reports Using Callbacks

Complete the following steps to add a logo to the header of the HTML report using a Report callback in the process model.

1. Open `<TestStand Public>\Tutorial\Computer.seq`.
2. Select **File»Save *<filename>* As** and save the sequence file as `Computer7.seq` in the `<TestStand Public>\Tutorial` directory.
3. Select **Configure»Result Processing** to launch the Result Processing dialog box.
4. Click the icon in the Options column for the built-in Report model plug-in to launch the Report Options dialog box and complete the following steps.
   a. On the Contents tab, select **HTML Document** from the Report Format ring control.
   b. Click **OK** to close the Report Options dialog box, and click **OK** to close the Result Processing dialog box.
5. Select **Edit»Sequence File Callbacks** to launch the Sequence File Callbacks dialog box, select the **ModifyReportHeader** callback, and click **Add** to add the callback to the sequence file.
6. Click **Edit** to close the Sequence File Callbacks dialog box and edit the new ModifyReportHeader callback sequence in the Sequence File window, which TestStand automatically opens.
7. Click the **Variables** pane, right-click the **Locals** item, select **Insert Local»String** from the context menu, and name the local variable `AddToHeader`. Click in the Value column of the AddToHeader variable and enter the following text:

   ```
   <img alt='Logo Goes Here' src='Logo.jpg'><br><br>
   <a href='http://www.ni.com'>Visit Our Website</a><br><br>
   ```
8. Insert a Statement step in the Main step group and rename the step `Add Custom Logo`.
9. On the Expression tab of the Step Settings pane, enter the following expression in the Expression control:

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

   Click the **Check Expression for Errors** button to verify that the expression contains valid syntax.
10. Save the changes and select **Execute»Single Pass**.
11. Click **Done** in the Test Simulator dialog box. Review the report after the execution completes and notice the new logo image at the top of the UUT report.
12. Complete the following steps to change the report settings back to the default settings.
    a. Select **Configure»Result Processing** to launch the Result Processing dialog box.
    b. Enable the **Show More Options** option.
    c. Click the **Reset to Defaults** button.
    d. Click **OK** in the Reset Configuration prompt.
    e. Click **OK** to close the Result Processing dialog box.
13. Close all the windows in the sequence editor.

11

Calling LabVIEW VIs

Use the LabVIEW Adapter to call VIs from TestStand, to create new LabVIEW projects and VIs to call from TestStand, to edit and debug existing VIs, and to use LabVIEW data types in TestStand.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Required LabVIEW Settings

All the tutorials in this chapter require you to 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.

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 in LabVIEW. By default, LabVIEW allows access to all VIs. Select **Tools»Options** to launch the Options dialog box. Select the **VI Server** category and browse to the Exported VIs section.
- Confirm that `"*"` is included in the VI Server: Exported VIs list and that the **Allow Access** option is enabled.

Creating and Configuring Steps with the LabVIEW Adapter

You can create steps that call stand-alone VIs, call VIs in the context of a LabVIEW project, and call LabVIEW class member VIs.

Calling a Stand-alone VI

Complete the following steps to insert a new LabVIEW step and configure the step to call a stand-alone VI.

1. Select **File»New»Sequence File** to open a new sequence file.
2. Save the sequence file as `Call LabVIEW VI.seq` in the `<TestStand Public>\Tutorial` directory.

3. Click the **LabVIEW** adapter icon, as shown in the following figure and located at the top of the Insertion Palette.

4. Insert a Pass/Fail Test step in the Main step group and rename the new step `LabVIEW Pass/Fail Test`.
5. On the LabVIEW Module tab of the Step Settings pane, click the **Browse for VI** button located to the right of the VI Path control, navigate to `<TestStand Public>\Tutorial\LabVIEW Pass-Fail Test.vi`, and click **Open**. 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.
6. In the VI Parameter Table, enter the values shown in Table 11-1.

Table 11-1. LabVIEW Pass/Fail Test Step Parameter Values

Parameter Name	Value
PASS/FAIL Flag	`Step.Result.PassFail`
Report Text	`Step.Result.ReportText`

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

Notice that TestStand automatically fills in the Value column of the **error out** output parameter with the `Step.Result.Error` property. By default, when a VI uses the standard LabVIEW **error out** cluster as an output parameter, TestStand automatically passes that value into the `Step.Result.Error` property for the step. You can also update the value manually. If an error occurs during execution of the VI and the **error out** cluster is passed to `Step.Result.Error`, TestStand launches the Run-Time Error dialog box by default.

7. Save the changes. Leave the sequence file open for the next tutorial.

Calling a VI in the Context of a LabVIEW Project

Complete the following steps to insert a new LabVIEW step and configure the step to call a VI in the context of a LabVIEW project.

1. Insert another Pass/Fail Test step in the Main step group and rename the new step `LabVIEW Project Pass/Fail Test`.
2. On the LabVIEW Module tab, click the **Browse for LabVIEW Project** button located to the right of the Project Path control, navigate to `<TestStand Public>\Tutorial\Test Project.lvproj`, and click **Open**. Notice that some of the TestStand button icons and tool tips change to indicate support for LabVIEW projects.

3. Click the **Browse for VI in LabVIEW Project** button, as shown in the following figure and located to the right of the VI Path control, to launch the Select VI from LabVIEW Project dialog box.

4. Select `LabVIEW Project Pass-Fail Test.vi` and click **OK**. Notice that the path defined in the project is selected as the VI path.
5. In the VI Parameter Table, enter the values shown in Table 11-2.

Table 11-2. LabVIEW Project Pass/Fail Test Step Parameter Values

Parameter Name	Value
PASS/FAIL Flag	`Step.Result.PassFail`
Report Text	`Step.Result.ReportText`

6. Save the changes. Leave the sequence file open for the next tutorial.

Calling LabVIEW Class Member VIs

Complete the following steps to insert two new LabVIEW steps, configure the first step to call a static member VI from a LabVIEW class to create a LabVIEW class object, and configure the second step to call a dynamically dispatched member method VI on the created object.

Note You must have LabVIEW 2012 or later to call member VIs from a LabVIEW class in TestStand and to use LabVIEW dynamic dispatching when calling LabVIEW classes in TestStand.

Calling a Static Member VI from a LabVIEW Class

Complete the following steps to insert a new LabVIEW step and configure the step to call a static member VI from a LabVIEW class in order to create a LabVIEW class object.

1. Insert an Action step in the Main step group and rename the new step `Create LabVIEW Class Object`.
2. On the LabVIEW Module tab, select **Class Member Call** from the Call Type ring control. Notice that some of the TestStand button icons and tool tips change to indicate support for LabVIEW class member calls.
3. Click the **Browse for LabVIEW Project** button, navigate to `<TestStand Public>\Tutorial\Test Project.lvproj`, and click **Open**.

4. Click the **Browse for LabVIEW Class in LabVIEW Project** button, as shown in the following figure and located to the right of the Class Path control, to launch the Select a Class from LabVIEW Project dialog box.

5. Select `LabVIEW Child Class.lvclass` and click **OK**.
6. Select **LabVIEW Child Static Factory Member.vi** from the Member Name ring control.
7. In the VI Parameter Table, enter `Locals.myChildObject` in the Value column of the **LabVIEW Child Class out** output parameter.

   The `myChildObject` local variable name appears in red text because it does not exist yet. You create this variable in the next step.
8. Right-click the `Locals.myChildObject` value and select **Create Locals.myChildObject»Object Reference** from the context menu to create `Locals.myChildObject` as an object reference local variable, which appears on the Variables pane.
9. Save the changes. Leave the sequence file open for the next tutorial.

Calling a Dynamically Dispatched Member Method VI

Complete the following steps to insert a new LabVIEW step and configure the step to call a dynamically dispatched member method VI on the LabVIEW class object that the Create LabVIEW Class Object step creates.

1. Insert another Pass/Fail Test step in the Main step group and rename the new step `LabVIEW Class Member Pass/Fail Test`.
2. On the LabVIEW Module tab, select **Class Member Call** from the Call Type ring control.
3. Click the **Browse for LabVIEW Project** button, navigate to `<TestStand Public>\Tutorial\Test Project.lvproj`, and click **Open**.
4. Click the **Browse for LabVIEW Class in LabVIEW Project** button to launch the Select a Class from LabVIEW Project dialog box.
5. Select `LabVIEW Parent Class.lvclass` and click **OK**.
6. Select **LabVIEW Dynamic Pass-Fail Test Member.vi** from the Member Name ring control.

7. In the VI Parameter table, enter the values shown in Table 11-3.

Table 11-3. LabVIEW Class Member Pass/Fail Test Step Parameter Values

Parameter Name	Value
LabVIEW Parent Class in	`Locals.myChildObject`
LabVIEW Parent Class out	`Locals.myChildObject`
PASS/FAIL Flag	`Step.Result.PassFail`
Report Text	`Step.Result.ReportText`

8. Save the changes. Leave the sequence file open for the next tutorial.

Executing the Sequence

Complete the following steps to execute the sequence of steps.

1. Select **Execute»Single Pass**. When the execution completes, the report indicates the steps passed.
2. Close the Execution window. Leave the sequence file open for the next tutorial.

Note You can run a VI multiple times before returning to TestStand. However, LabVIEW passes the results from only the most recent run to TestStand when you finish debugging.

Creating and Editing VIs

You can use the LabVIEW Adapter to create a new VI from TestStand, create a new LabVIEW project and add a new VI to the project from TestStand, and edit an existing VI from TestStand.

Creating a New VI from TestStand

Complete the following steps to create a new VI from TestStand.

1. Select **File»Save *<filename>* As** and save the sequence file as `Call LabVIEW VI 2.seq` in the `<TestStand Public>\Tutorial` directory.
2. Insert a Numeric Limit Test step after the LabVIEW Class Member Pass/Fail Test step and rename the new step `LabVIEW Numeric Limit Test`.
3. On the LabVIEW Module tab, complete the following steps to configure the LabVIEW Numeric Limit Test step.
    a. Click the **Create VI** button, as shown in the following figure and located to the right of the VI Path control, to create a new VI.

b. In the File dialog box, browse to the `<TestStand Public>\Tutorial` directory, enter `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 the 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 whether the step passes or fails. Code modules can pass step properties as parameters to and from the code module or use the TestStand API in the code module to update step properties. When you use a default code template from National Instruments to create a code module, TestStand creates the parameters needed to access the step properties 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 enter `10.0`.

e. Save and close the VI.

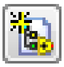
4. In TestStand, click the **LabVIEW Module** tab. Notice that TestStand automatically updates the output parameters for the VI based on the information stored in the code template for the Numeric Limit Test step type.
5. Save the changes and select **Execute»Single Pass**. When the execution completes, the report indicates the step passed with a numeric measurement of `10`.
6. Close the Execution window. Leave the sequence file open for the next tutorial.

Creating a New LabVIEW Project and Adding a New VI to the Project from TestStand

Complete the following steps to create a new LabVIEW project and add a new VI to that project from TestStand.

1. Insert another Numeric Limit Test step after the LabVIEW Numeric Limit Test step and rename the new step `LabVIEW Project Numeric Limit Test`.
2. On the LabVIEW Module tab, complete the following steps to configure the LabVIEW Project Numeric Limit Test step.

a. Click the **Create LabVIEW Project** button, as shown in the following figure and located to the right of the Project Path control, to create a new LabVIEW Project.



b. In the File dialog box, browse to the `<TestStand Public>\Tutorial` directory, enter `LabVIEW Project Test.lvproj` in the File Name control, and click **OK**. TestStand creates a new LabVIEW Project and opens the project in LabVIEW.

c. In TestStand, click the **Add or Remove VIs from LabVIEW Project** button, as shown in the following figure and located to the right of the VI Path control, to launch the Add or Remove Items from LabVIEW Project dialog box.

d. Right-click the **My Computer** item in the LabVIEW Project control and select **New VI** from the context menu to launch the Select the LabVIEW VI to Create dialog box.

e. In the Select LabVIEW VI to Create dialog box, browse to the `<TestStand Public>\Tutorial` directory, enter `LabVIEW Project 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 step type and adds the VI to the LabVIEW project.

f. Right-click the new VI under the My Computer item and select **Edit VI** from the context menu. TestStand opens the VI in LabVIEW within the project context, which LabVIEW displays in the bottom left corner of the front panel.

g. In LabVIEW, open the block diagram for the VI.

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

i. Save and close the VI.

3. In TestStand, select `LabVIEW Project Numeric Limit Test.vi` in the Add or Remove Items from the LabVIEW Project dialog box, and click **Select** to close the Add or Remove Items from LabVIEW Project dialog box.

   On 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 step type.

4. Save the changes and select **Execute»Single Pass**. When the execution completes, the report indicates the steps passed with a numeric measurement of `10.0`.

5. Close the Execution window. Leave the sequence file open for the next tutorial.

Editing Existing VIs from TestStand

Complete the following steps to edit existing VIs from TestStand.

1. Select the **LabVIEW Pass/Fail Test** step and use the LabVIEW Module tab to complete the following steps.

   a. Click the **Edit VI** button, as shown in the following figure and located to the right of the VI Path control. LabVIEW becomes the active application in which the LabVIEW Pass-Fail Test VI is open.

b. Open the block diagram for the VI and change the **PASS/FAIL Flag** Boolean constant to `False`.

c. Save and close the VI.

2. In TestStand, select the **LabVIEW Project Pass/Fail Test** step and use the LabVIEW Module tab to complete the following steps.
   a. Click the **Edit VI** button. LabVIEW becomes the active application and the LabVIEW Project Pass-Fail Test VI is open within the LabVIEW project context.
   b. Open the block diagram for the VI and change the **PASS/FAIL Flag** Boolean constant to `False`.
   c. Save and close the VI.
3. In TestStand, save the changes and select **Execute»Single Pass**. When the execution completes, the report indicates the LabVIEW Pass/Fail Test and LabVIEW Project Pass/Fail Test steps failed.
4. Close the Execution window. Leave the sequence file open for the next tutorial.

Debugging VIs

Complete the following steps to debug VIs you call from TestStand using the LabVIEW Adapter.

1. Place a breakpoint at the LabVIEW Pass/Fail Test step.
2. Select **Execute»Run MainSequence**. The execution pauses at the LabVIEW Pass/Fail Test step.
3. Complete the following steps to debug the LabVIEW Pass-Fail Test VI the LabVIEW Pass/Fail Test step calls.
   a. Click the **Step Into** button on the Debug toolbar in TestStand. LabVIEW becomes the active application, in which the LabVIEW Pass-Fail Test VI is open and in a suspended state.
   b. Open the block diagram of the suspended VI.
   c. Click the **Step Into** or **Step Over** button on the LabVIEW toolbar to begin stepping through the VI. You can click the **Finish VI** button at any time to finish stepping through the VI.
   d. When you finish stepping through the VI, click the **Return to Caller** button on the LabVIEW toolbar to return to TestStand. The execution pauses at the next step in the sequence.
4. Click the **Resume** button on the Debug toolbar in TestStand to complete the execution.
5. Close the Execution window.
6. Remove the breakpoint from the LabVIEW Pass/Fail Test step. Leave the sequence file open for the next tutorial.

Creating TestStand Data Types from LabVIEW Clusters

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

Complete the following steps to create a TestStand data type that matches a LabVIEW cluster.

1. Select **File»Save <*filename*> As** and save the sequence file as `Call LabVIEW VI 3.seq` in the `<TestStand Public>\Tutorial` directory.
2. Insert a Pass/Fail Test step in the Main step group after the LabVIEW Project Numeric Limit Test step and rename the new step `Pass Container to VI`.
3. On the LabVIEW Module tab, click the **Browse for VI** button, navigate to `<TestStand Public>\Tutorial\VI with Cluster Input.vi`, and click **Open**.
4. Click the **Create/Update Custom Data Type** button, as shown in the following figure and located in the Type column of the **Input Cluster** parameter in the VI Parameter Table, to launch the Create/Update 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.
5. In the Create/Update Custom Data Type From Cluster dialog box, change the type name to `InputData` and click the **Create** button to accept the automatically assigned values and to create the data type in the current sequence file.
6. On the LabVIEW Module tab, remove the checkmark from the Default column for the **Input Cluster** input parameter, click the **Expression Browse** button in the Value column to launch the Expression Browser dialog box, and complete the following steps.
   a. On the Variables/Properties tab, right-click the **Locals** item and select **Insert Types» InputData** to create a local variable of the InputData data type. Rename the local variable `ContainerData`.
   b. Right-click the `Number` subproperty of `ContainerData` and select **Properties** from the context menu to launch the Number Properties dialog box.
   c. Enter `23` in the **Value** control and click **OK**.
   d. Right-click the `String` subproperty of `ContainerData` and select **Properties** from the context menu to launch the String Properties dialog box.

   e. Enter `My String Data` in the **Value** control and click **OK**.
   f. Enter `Locals.ContainerData` in the **Expression** control on the Variables/Properties tab and click **OK**. The Value column for the **Input Cluster** parameter now contains `Locals.ContainerData`.

7. Enter `Step.Result.ReportText` in the **Value** column for the **Report Text** 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 **Number** and **String** elements of the **Input Cluster** parameter to the `ReportText` property of the step.
8. Save the changes and select **Execute»Single Pass**. When the execution completes, the report shows the text the VI with Cluster Input VI returns.
9. Close all the windows in the sequence editor.

12

Calling LabWindows/CVI Code Modules

Use the LabWindows/CVI Adapter to call LabWindows/CVI code modules from TestStand, to create new code modules to call from TestStand, to edit and debug existing code modules, and to use LabWindows/CVI data types in TestStand.

Refer to the *NI TestStand Help* for more information about all the features covered in this chapter.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\Solution` directory.

Required LabWindows/CVI Settings

All the tutorials in this chapter require you to have the LabWindows/CVI development system and TestStand installed on the same computer.

In addition, complete the following steps to configure the LabWindows/CVI Adapter to execute steps in an external instance of the LabWindows/CVI development system and to allow only new code templates.

1. Select **Configure»Adapters** to launch the Adapter Configuration dialog box.
2. Select **LabWindows/CVI** in the Adapter column and click the **Configure** button to launch the LabWindows/CVI Adapter Configuration dialog box.
3. In the Step Execution section, select the **Execute Steps in an External Instance of CVI** option.
4. In the Code Template Policy section, select the **Allow Only New Templates** option.
5. Click **OK** to close the LabWindows/CVI Adapter Configuration dialog box, click **OK** again when the adapter displays a warning that confirms TestStand will unload all modules, and click **Done** in the Adapter Configuration dialog box.

Creating and Configuring Steps with the LabWindows/CVI Adapter

Complete the following steps to insert a new step that uses the LabWindows/CVI Adapter and configure the step to call a code module.

1. Select **File»New»Sequence File** to open a new sequence file.
2. Save the sequence file as `CallCVICodeModule.seq` in the `<TestStand Public>\Tutorial` directory
3. Click the **LabWindows/CVI** adapter icon at the top of the Insertion Palette.
4. Insert a Pass/Fail Test step in the Main step group and rename the new step `CVI Pass/Fail Test`.
5. On the LabWindows/CVI Module tab of the Step Settings pane, click the **File Browse** button located to the right of the Module control, navigate to `<TestStand Public>\Tutorial\CallCVICodeModule.dll`, and click **Open**.
6. Select **PassFailTest** from the Function ring control.

Note When you select a function, the LabWindows/CVI Adapter attempts to read the export information LabWindows/CVI includes in the DLL or the function parameter information from the type library in the code module, if one exists. When the function parameter information is not defined, you can select a code template from the Code Template ring control to specify the function prototype or add parameters to the Parameters Table control to specify the function prototype.

7. Select **PassFail template for LabWindows/CVI** from the Code Template ring control. The Parameters Table control contains the default value expressions the code template specifies. When TestStand calls the code module, the LabWindows/CVI Adapter stores the returned values for the result and the error details in the specified properties of the step.
8. Save the changes and select **Execute»Single Pass**. Because the LabWindows/CVI Adapter is configured to use an external instance of LabWindows/CVI to execute code modules, TestStand launches the LabWindows/CVI development environment to execute the function the step calls. When the execution completes, the report indicates the step passed.
9. Select **File»Unload All Modules** to unload the DLL the step calls so you can rebuild the DLL later in this chapter.
10. Close the Execution window. Leave the sequence file open for the next tutorial.

Creating a New Code Module from TestStand

Note National Instruments recommends using DLL files when you develop code modules using the LabWindows/CVI Adapter. The tutorials in this manual demonstrate creating and debugging only DLL code modules.

Complete the following steps to create a new code module from TestStand.

1. Select **File»Save <*filename*> As** and save the sequence file as `CallCVICodeModule2.seq` in the `<TestStand Public>\Tutorial` directory.
2. Insert a Numeric Limit Test step after the CVI Pass/Fail Test step and rename the new step `CVI Numeric Limit Test`.
3. On the LabWindows/CVI Module tab, complete the following steps to configure the CVI Numeric Limit Test step.
   a. Click the **File Browse** button, navigate to `<TestStand Public>\Tutorial\CallCVICodeModule.dll`, and click **Open**.
   b. Enter `NumericLimitTest` in the Function ring control.
   c. Select **NumericLimit template for LabWindows/CVI** from the Code Template ring control. Notice that TestStand automatically updates the function prototype and parameter values based on the information stored in the code template for the Numeric Limit Test step type.
4. Click the **Source Code Files** button, as shown in the following figure and located to the right of the Parameter Details Table control, to launch the CVI Source Code Files window and complete the following steps.
   a. Enter `CVINumericLimitTest.c` in the Source File Containing Function control.
   b. For the CVI Project File to Open control, click the **File Browse** button, navigate to `<TestStand Public>\Tutorial\CallCVICodeModule.prj`, and click **Open**.
   c. Click **Close**.
5. Click the **Create Code** button, as shown in the following figure and located just below the **Source Code Files** button, to create a code module.

   When you click the **Create Code** button, TestStand launches the Select a Source File dialog box, in which you can specify the source code to use for the code module.

6. Navigate to the `<TestStand Public>\Tutorial` directory and click **OK**. TestStand creates a new code module source file named `CVINumericLimitTest.c` based on the available code templates for the TestStand Numeric Limit Test step type and opens the code module in LabWindows/CVI.

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 whether the step passes or fails. Code modules can pass step properties as parameters to and from the code module or use the TestStand API in the code module to update step properties. When you use a default code template from National Instruments to create a code module, TestStand creates the parameters needed to access the step properties for you.

7. In LabWindows/CVI, uncomment the following code in the source file:

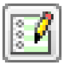
```
double testMeasurement = 10.0;
double lowLimit;
*measurement = testMeasurement;
```

8. Save and close the source file. Leave LabWindows/CVI open.
9. In the LabWindows/CVI project window, select **Build»Create Debuggable Dynamic Link Library** to rebuild the DLL. Click **OK** when LabWindows/CVI prompts you that the files are created.
10. In TestStand, save the changes and select **Execute»Single Pass**. When the execution completes, the report indicates the step passed with a numeric measurement of `10`.
11. Select **File»Unload All Modules** to unload the DLL.
12. Close the Execution window. Leave the sequence file and LabWindows/CVI open for the next tutorial.

Editing an Existing Code Module from TestStand

Complete the following steps to edit an existing code module from TestStand.

1. Select the **CVI Numeric Limit Test** step and use the LabWindows/CVI Module tab to complete the following steps.
    a. Click the **Edit Code** button, as shown in the following figure and located just below the **Create Code** button. LabWindows/CVI becomes the active application in which the `CVINumericLimitTest.c` source file is open.



    b. Change the initial value in the declaration for the testMeasurement variable to `5.0`.
    c. Save and close the source file. Leave LabWindows/CVI open.
2. Rebuild the DLL. Click **OK** when LabWindows/CVI prompts you that the files are created.

3. In TestStand, select **Execute»Single Pass**. When the execution completes, the report indicates the step failed with a numeric measurement of `5`.
4. Close the Execution window. Leave the sequence file and LabWindows/CVI open for the next tutorial.

Debugging a Code Module

Complete the following steps to debug a code module you call from TestStand using the LabWindows/CVI Adapter.

1. Place a breakpoint at the CVI Pass/Fail Test step.
2. Select **Execute»Run MainSequence**. The execution pauses at the CVI Pass/Fail Test step.
3. Complete the following steps to debug the code module the CVI Pass/Fail Test step calls.
   a. Click the **Step Into** button on the Debug toolbar in TestStand. LabWindows/CVI becomes the active application, in which the LabWindows/CVI Pass-Fail Test code module is open and in a suspended state.
   b. Click the **Step Into** button or the **Step Over** button on the LabWindows/CVI toolbar to begin stepping through the code module.
   c. When you finish stepping through the code module, click the **Finish Function** button on the LabWindows/CVI toolbar to return to TestStand. The execution pauses at the next step in the sequence.
4. Click the **Resume** button on the Debug toolbar in TestStand to complete the execution.
5. Close the Execution window.
6. Select **File»Unload All Modules** to unload the DLL.
7. Remove the breakpoint from the CVI Pass/Fail Test step. Leave the sequence file and LabWindows/CVI open for the next tutorial.

Creating TestStand Data Types from LabWindows/CVI Structs

TestStand provides number, string, Boolean, and object reference built-in data types. TestStand also provides several standard named data types, including Path and Error. You can create container data types to hold any number of other data types. TestStand container data types are analogous to C structures in LabWindows/CVI.

Complete the following steps to create a TestStand data type that matches a LabWindows/CVI struct and call a function in a DLL that has the struct as a parameter.

Creating a New Custom Data Type

Complete the following steps to create a new container data type that contains numeric and string subproperties.

1. Click the **Types** button on the Environment toolbar to open the Types window.
2. Select the `CallCVICodeModule2.seq` sequence file on the View Types For pane.
3. Right-click the **Custom Data Types** item and select **Insert Custom Data Type» Container** from the context menu to insert a new data type. Rename the new container data type `CVITutorialStruct`.
4. Expand the `CVITutorialStruct` item.
5. Right-click the `CVITutorialStruct` item and select **Insert Field»Number** from the context menu to insert a new field in the data type. Rename the new field `Measurement`.
6. Right-click the `CVITutorialStruct` item and select **Insert Field»String** from the context menu to insert another new field in the `CVITutorialStruct` container data type. Rename the new field `Buffer`.
7. Save the changes. If TestStand warns you that the sequence file contains modified types, review the information in the warning dialog box, select the **Increment Type Versions** option, and click **OK**. Leave the sequence file open for the next tutorial.

Specifying Structure Passing Settings

Complete the following steps to specify the structure passing properties for the `CVITutorialStruct` container data type.

1. Right-click the `CVITutorialStruct` item in the Types window and select **Properties** from the context menu to launch the Type Properties dialog box. The name of the Type Properties dialog box is specific to the name of the property you select.
2. Click the **C Struct Passing** tab of the Type Properties dialog box.
3. Enable the **Allow Objects of This Type to be Passed as Structs** option. The Property ring control lists the two fields in the `CVITutorialStruct` container data type. Notice that the Numeric Type control for the Measurement property defaults to 64-bit Real Number (double).
4. Select the **Buffer** property from the Property ring control.
5. For the String Type control, select the **C String Buffer** option to allow the C function to alter the value of the structure field.
6. Click the **Version** tab of the Type Properties dialog box and disable the **Modified** option.
7. Select **OK** to close the Type Properties dialog box.
8. Save the changes. Close the Types window. Leave the sequence file open for the next tutorial.

Calling a Function with a Struct Parameter

Complete the following steps to use the `CVITutorialStruct` container data type as a parameter to a function a step calls.

1. Select **File»Save <*filename*> As** and save the sequence file as `CallCVIModule3.seq` in the `<TestStand Public>\Tutorial` directory.
2. Click the **Variables** pane.
3. Right-click the **Locals** item, select **Insert Local»Type»CVITutorialStruct** from the context menu to insert an instance of the container data type, and rename the new variable `CVIStruct`.
4. Insert an Action step into the Main step group after the CVI Numeric Limit Test step and rename the new step `Pass Struct Test`.
5. On the LabWindows/CVI Module tab, click the **File Browse** button, navigate to `<TestStand Public>\Tutorial\CallCVICodeModule.dll`, and click **Open**.
6. Enter `PassStructTest` in the Function control.
7. Click the **Add Parameter** button, as shown in the following figure and located to the right of the Parameters Table control, to insert a new parameter in the Parameters Table control.

8. Enter the following information in the Parameter Details Table control:
   a. In the **Name** field, rename the parameter `cviStruct`.
   b. In the **Category** field, select `C Struct`.
   c. In the **Type** field, select `CVITutorialStruct`.
9. Enter `Locals.CVIStruct` in the Value Expression column for the parameter in the Parameters Table control.
10. Click the **Source Code Files** button to launch the CVI Source Code Files window. Complete the following steps to select the source file and project file to use when inserting the function.
    a. Enter `CVIStructPassingTest.c` in the Source File Containing Function control.
    b. For the CVI Project File to Open control, click the **File Browse** button, navigate to `<TestStand Public>\Tutorial\CallCVICodeModule.prj`, and click **Open**.
    c. Click **Close**.
11. Click the **Create Code** button to create a code module. TestStand launches the Select a Source File dialog box.
12. Browse to the `<TestStand Public>\Tutorial` directory and click **OK**. TestStand creates a new source file named `CVIStructPassingTest.c` with an empty function.

13. In LabWindows/CVI, complete the following steps.
    a. Add the following type definition before the `PassStructTest` function:

    ```
    struct CVITutorialStruct {
        double measurement;
        char buffer[256];
    };
    ```

    b. Add the following code to the `PassStructTest` function:

    ```
    if (cviStruct)
    {
        cviStruct->measurement = 10.0;
        strcpy(cviStruct->buffer, "Average Voltage");
    }
    ```

    c. Add the following statement to the top of the source file to include the declaration of the `strcpy` function:

    ```
    #include <ansi_c.h>
    ```

14. Save and close the source file.
15. In the LabWindows/CVI project window, select **Build»Create Debuggable Dynamic Link Library** to rebuild the DLL. Click **OK** when LabWindows/CVI notifies you that the files are created.
16. In TestStand, place a breakpoint at the new Pass Struct Test step.
17. Save the changes and select **Execute»Run MainSequence**.
18. Step through the sequence and review the values in the `Locals.CVIStruct` variable on the Variables pane of the Execution window before and after executing the Pass Struct Test step.
19. Select **File»Unload All Modules** to unload the DLL.
20. Remove the breakpoint from the CVI Pass/Fail Test step. Close all the windows in the sequence editor.

A

Technical Support and Professional Services

Log in to your National Instruments `ni.com` User Profile to get personalized access to your services. Visit the following sections of `ni.com` for technical support and professional services:

- **Support**—Technical support at `ni.com/support` includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit `ni.com/support` 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. Registered users also receive access to the NI Discussion Forums at `ni.com/forums`. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support, as well as exclusive access to self-paced online training modules at `ni.com/self-paced-training`. All customers automatically receive a one-year membership in the Standard Service Program (SSP) with the purchase of most software products and bundles including NI Developer Suite. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit `ni.com/ssp` for more information.

    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 training and certification program information. You can also 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`.

You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index