RELEASE NOTES

TestStand™

Version 3.5

These release notes contain system requirements for TestStand 3.5, as well as information about new features, documentation resources, and other changes since TestStand 3.1.

Contents

NATIONAL INSTRUMENTS™

Minimum System Requirements

To run TestStand 3.5, National Instruments recommends that your system meet the following requirements:

- Windows 2000/XP
- 600 MHz Pentium class microprocessor (400 MHz minimum)
- 128 MB of memory (256 MB recommended)
- 500 MB of free hard disk space (255 MB minimum)

Note During installation, TestStand requires additional free hard disk space to uncompress files.

- SVGA resolution or higher video adapter, with a minimum 800 × 600 video resolution for small fonts or a minimum 1,024 × 768 for large fonts
- Microsoft Internet Explorer version 6.0 or later (5.5 or later required)

TestStand 3.5 is compatible with the following National Instruments application development environments (ADEs):

- LabVIEW 6.1 or later. LabVIEW 7.0 or later is required to use the TestStand User Interface Controls. LabVIEW 7.1 or later is required to configure and call LabVIEW Express VIs.

Note TestStand 3.5 and earlier is not compatible with the LabVIEW 8.0 Real-Time Module. TestStand 3.5 only supports the LabVIEW 7.1.1 Real-Time Module and earlier.

- LabWindows™/CVI™ 6.0 or later. LabWindows/CVI 7.0 or later is recommended for use with the TestStand User Interface Controls.
- Measurement Studio 7.0 (or later) Enterprise Edition is required for integration with Microsoft Visual Studio .NET 2003 or later.

Installing Multiple Versions of TestStand on the Same Computer

You cannot install TestStand 3.5 over a previous version of TestStand. In order to install TestStand into a directory that contains a previous version, you must first uninstall the previous version. However, since the uninstallers for versions of TestStand prior to 3.0 remove the `<TestStand>\OperatorInterfaces\User` directory, you must use the following procedure to safely uninstall the previous version of TestStand and preserve all configuration files and files located in the `User` subdirectories:

1. Move the `<TestStand>\OperatorInterfaces\User` directory to a location on your computer that is outside the `<TestStand>` directory.
2. Run the TestStand Uninstaller by selecting **Start»Control Panel» Add/Remove Programs»National Instruments Software**.

   The TestStand Uninstaller might launch a dialog box that requests confirmation to remove all TestStand configuration files and pre-installed user components. Click **No**.
3. When the TestStand Uninstaller completes, move the `OperatorInterfaces\User` directory back into the original `<TestStand>` directory.

   You can now install TestStand 3.5 into this directory.

Although TestStand 3.5 will install on a machine that contains a previous TestStand version, only one version of TestStand can be active at a time. If you must install TestStand 3.5 on the same machine as an earlier TestStand version, you can use the TestStand Version Selector to specify the active version of TestStand. To launch the TestStand Version Selector, select **Start»Programs»National Instruments»TestStand 3.5» TestStand Version Selector**. The TestStand Version Selector application, `TSVerSelect.exe`, is located in the following directory: `C:\Program Files\National Instruments\Shared\TestStand Version Selector`.

If you activate TestStand 3.5 and run an operator interface from your previous TestStand installation, the operator interface uses the engine, step types, and components from TestStand 3.5. If you activate your previous installation of TestStand and run a TestStand 3.5 Operator Interface or the TestStand Sequence Editor, those applications will not function correctly.

Migrating User Components

You can copy the following directories and files from your previous TestStand installation to the TestStand 3.5 installation directory:

- `<TestStand>\Components\User`
- `<TestStand>\CodeTemplates\User`
- `<TestStand>\OperatorInterfaces\User`
- `<TestStand>\Cfg\TypePalettes\MyTypes.ini`
- `<TestStand>\Cfg\SeqEdit.ini`
- `<TestStand>\Cfg\TestExec.ini`
- `<TestStand>\Cfg\TestStandDatabaseOptions.ini`
- `<TestStand>\Cfg\TestStandModelModelOptions.ini`
- `<TestStand>\Cfg\TestStandModelReportOptions.ini`
- `<TestStand>\Cfg\Users.ini`

If you have custom Tools menu items in your previous TestStand installation, use the following procedure to export the items from that installation and then import them into TestStand 3.5:

1. In your previous installation of TestStand, launch the Customize Tools Menu dialog box from the TestStand Sequence Editor and click **Export Items to File** to launch the Export Tools Menu dialog box.
2. In the Export Tools Menu dialog box, select the menu items to export to a Tools menu file.
3. Create the following directory under TestStand 3.5:

   `<TestStand>\Setup\ToolMenusToInstall`
4. Place the Tools menu file in the new directory.
5. Launch the TestStand 3.5 Sequence Editor to allow TestStand to add the new menu items to the Tools menu and then delete the Tools menu file.

Migrating Your Changes to TestStand Components

TestStand includes several components that you can customize, such as process models, operator interfaces, and certain step types. If you have made changes to one of these components and placed the changes in the appropriate user directory, TestStand will not overwrite your changes when you install TestStand 3.5. Your modified component will continue to function correctly with TestStand.

To keep the changes you have made to a component and incorporate the new functionality provided in TestStand 3.5, select one of the following options:

- **If you have made substantial or complex changes to the component**—Use a file-differencing tool to determine the changes between the TestStand 3.5 version of the component and the original version of the component that you modified. Then, apply the TestStand 3.5 improvements to your version of the component.
- **If you have made minor changes to the component**—Use a file-differencing tool to determine the changes you made to the component. Reapply your improvements to a copy of the TestStand 3.5 version of the component.

The following types of file-differencing tools are available:

- To compare sequence files, use the TestStand Differ, which you access from the TestStand Sequence Editor by selecting **Edit»Diff Sequence File With**.
- To compare text files, use a source code differencing tool such as Microsoft Windiff, or the Diff command, which is located in the Edit menu of the Source and Execution window in LabWindows/CVI.
- To compare VI files, use the Compare VIs tool in LabVIEW by selecting **Tools»Compare**. This tool is only available if you have the LabVIEW Professional Development System installed.

Note Subsets of different versions of the same component are not necessarily interoperable without modifications. For example, you cannot replace a single sequence in the TestStand 3.5 process models with the corresponding sequence from older TestStand process models without making further modifications. If you customized the TestStand process models, you must ensure that all subordinate components used by the process models can be found and that any of those components that are ActiveX servers are registered. The main process model sequence for TestStand uses separate sequences, DLLs, and ActiveX servers to support database logging and report generation features.

TestStand and Windows XP Service Pack 2

Windows XP Service Pack 2 has added security measures that affect remote sequence calls in TestStand. For more information about setting up TestStand as a server for remote execution and remote sequence calls, refer to the *TestStand 3.0 Documentation Updates* section and the following KnowledgeBase article: *TestStand Remote Execution Fails When Using Windows XP Service Pack 2 With Error -17850 or -17851*.

Behavior Changes

TestStand includes the following behavior changes between version 3.1 and version 3.5.

- The TestStand Deployment Utility no longer uses the Keywords, Author, Subject, and Comments options found in the Advanced Installer Options dialog box.
- To select an instance VI within a polymorphic VI, you must configure the LabVIEW Adapter to use the LabVIEW development system in the LabVIEW Adapter Configuration dialog box. Previous versions of TestStand allowed you to select a polymorphic VI when the LabVIEW Adapter was configured to use the LabVIEW Run-Time Engine.

  Alternatively, you can select the instance VI directly when the LabVIEW Adapter is configured to use the LabVIEW Run-Time Engine.

Refer to the *Using LabVIEW 8.0* section of this document for any additional changes in behavior for TestStand in support of LabVIEW between previous versions of LabVIEW and LabVIEW 8.0.

What's New in TestStand 3.5

This section describes the new features in TestStand 3.5 and other changes since TestStand 3.1.

Requirements Management Support

TestStand 3.5 includes built-in fields for notating product and unit requirements. You can specify requirements information at the following levels: workspace, project, sequence file, sequence, and step.

The format of the string values that represent requirements can conform to third-party requirements management packages. In addition, requirements management software packages can use the TestStand API to retrieve or specify requirement values directly.

TestStand Deployment Utility Enhancements

In TestStand 3.5, the TestStand Deployment Utility allows you to create installers that include NI drivers, hardware configuration files, and NI software components for distribution.

Note The size of installed software remains unchanged when you deploy a TestStand system. Only the size of the installer increases.

Note Additional NI installer(s) you select to include in your installer contain only the features currently installed on the build computer and, therefore, may not be complete copies of the original product(s). For more information, refer to the *TestStand Help*.

ATML Test Results Integration

TestStand 3.5 introduces the ability to generate XML reports that validate against the Test Results schema defined in the emergent Automated Test Result Markup Language (ATML) standard.

Because the Test Results schema is not yet finalized, generated XML reports comply with the release candidate version of the Test Results schema.

Memory Corruption Detection

TestStand 3.5 provides you with the following options to monitor memory when calling code modules:

- Detection of incorrectly altered thread stack contents when TestStand calls a DLL or ActiveX/COM DLL code module.
- Detection of modifications to memory surrounding buffers passed to DLL code modules.
- Display a list of top-level PropertyObjects with unreleased references during shutdown.

Password-Protected Sequence Files

TestStand 3.5 adds the ability to password-protect and restrict viewing of sequence files in the sequence editor and operator interfaces.

User Groups

In TestStand 3.5, user profiles have been changed to user groups. You can specify which groups each user belongs to. Users are granted the privileges of the groups they belong to, as well as the user-specific privileges you specify.

Message Popup Step Type Enhancements

TestStand 3.5 includes the following customization options in the enhanced Message Popup step type:

- **Formatting**—Ability to change the background color of dialog boxes and buttons. Also includes the capability to change the size, style, and color of text in dialog boxes.
- **Images**—Ability to display graphics (`.gif`, `.bmp`, `.ico`, `.jpg`, `.jpeg`, `.png`) or web files (`.htm`, `.html`, `.xhtml`, `.shtml`, `.mht`, `.mhtml`, `.txt`, `.xml`, `.asp`) in the dialog box.
- **Layout**—Ability to change the layout of controls on the dialog box. You can also create a floating dialog box with respect to the main window.

Call Executable Step Type Enhancements for Remote Execution

TestStand 3.5 includes the following enhancements to the Call Executable step type which allow you to run an executable remotely:

- Allows you to specify a remote host on which to call the executable.
- Includes an executable that you run on the remote host and waits for a connection from the Call Executable step type.

LabVIEW Adapter Support for Partial Cluster Passing

TestStand 3.5 adds the ability to specify whether you pass default values for individual elements of cluster controls that are recommended or optional. Previous versions of TestStand only allowed you to enable the Default checkbox for the entire cluster. In TestStand 3.5, the Specify Module dialog box displays a Default checkbox for each element of a cluster control. TestStand only requires you to specify an expression for elements of the cluster when the Default checkbox is disabled.

TestStand API Enhancements

The following table lists the new properties and methods that have been added to the TestStand API in TestStand 3.5.

Table 1. New TestStand API Properties and Methods

Class	Properties	Methods
Engine	PersistConfigFile	DisplayLockUnlockDialog GetUserGroup
LabVIEWAdapter	—	DeployProjectLibrary
LabVIEWParameter	ArrayElementPrototype DisplayType	—

Table 1. New TestStand API Properties and Methods (Continued)

Class	Properties	Methods
LabVIEWParameterElement	ArrayElementPrototype DisplayType	—
PropertyObjectFile	Protection Locked (Read Only)	Lock Unlock
StationOptions	DebugOptions	—
User	Members	—
UsersFile	UserGroupList UserList	—

Refer to the *TestStand Help* for more information about the classes, properties, and methods listed in Table 1.

Using LabVIEW 8.0

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

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

LabVIEW 8.0 Real-Time Module Incompatibility

TestStand 3.5 and earlier is not compatible with the LabVIEW 8.0 Real-Time Module. TestStand 3.5 only supports the LabVIEW 7.1.1 Real-Time Module and earlier.

Projects

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

Project Libraries

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

Network-Published Shared Variables

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

Deploying Variables

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

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

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

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

Using an Aliases File

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

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

- **LabVIEW Run-Time Engine**—LabVIEW looks for an aliases file with the same name and location as the TestStand application. For example, `SeqEdit.aliases` for the TestStand Sequence Editor and `TestExec.aliases` for an operator interface.
- **Development System**—LabVIEW looks for an aliases file located in the same directory as the LabVIEW development system executable.
- **Other Executable**—LabVIEW looks for an aliases file with the same name and directory as the LabVIEW executable server. For example, `TestStandLVRTS.aliases` for `TestStandLVRTS.exe` and `TestExec.aliases` for an operator interface.

Project DAQmx Tasks, Channels, and Scales

TestStand 3.5 does not support VIs that use DAQmx tasks, channels, and scales that are defined in a project. You must define your DAQmx tasks, channels, and scales in Measurement & Automation Explorer (MAX).

Conditional Disable Structures and Symbols

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

64-Bit Integer Data Type

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

User Access to VI Server

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

XControls

TestStand supports calling VIs that use XControls on Windows systems.

Remote Execution

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

Building a TestStand Deployment with LabVIEW 8.0

TestStand 3.5 supports deploying LabVIEW 8.0 VIs and VIs from project libraries. However, the following new restrictions exist that do not apply to earlier versions of LabVIEW:

- TestStand does not support deployment of duplicate project libraries. You should not attempt to include two project libraries with the same name.
- TestStand no longer supports including two VIs with the same name, unless the VIs are included in different project libraries.
- National Instruments does not recommend distributing two VIs with the same name to different locations on a target machine. If LabVIEW attempts to load a subVI, when a VI with the same name but from a different location is already in memory, LabVIEW uses the VI in memory. However, LabVIEW reports an error if you attempt to load a top-level VI, even if the similarly named VI in memory is an exact copy of the one you want to load.
- TestStand no longer supports distributing duplicate DLLs that are called by VIs. Note that TestStand 3.5 still supports, but does not recommend, distributing duplicate DLLs that are called by steps in a sequence file.
- National Instruments does not recommend editing packaged VIs on a deployed system because unexpected errors can occur. For example, TestStand only includes required VIs from a project library in a deployment. If you attempt to add new VIs from a project library to the

deployment image, the new VIs may not be found by LabVIEW when executed. Analysis and instrument drivers are examples of VIs that use project libraries.

National Instruments recommends that you rebuild the deployment image on the development system and redeploy it on the deployment system.

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

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

Using Microsoft Visual Studio .NET 2003 and Later

Supported Versions

The C/C++ DLL Adapter and the .NET Adapter in TestStand allow you to call DLL modules and .NET assemblies built with Microsoft Visual Studio .NET 2003. Additionally, if you have Measurement Studio 7.0 (or later) Enterprise Edition and Microsoft Visual Studio .NET 2003 installed, you can create, edit, and step into code modules directly from TestStand.

Microsoft Visual Studio 2005 was not released at the time TestStand 3.5 was released. Because of this, you might notice the following behaviors when using TestStand 3.5 with Microsoft Visual Studio 2005:

- TestStand 3.5 can successfully call DLLs built by Microsoft Visual Studio 2005.
- TestStand 3.5 can successfully call .NET assemblies built with .NET Framework 2.0. The .NET Framework 2.0 is available with Microsoft Visual Studio 2005.
- TestStand 3.5 does not support any integration features with Microsoft Visual Studio 2005 such as creating, editing, and stepping into code modules.

When more than one version of the .NET Framework is installed on a computer, an application can only load one version of the .NET run-time into memory. For unmanaged applications like the TestStand Sequence Editor, the .NET Adapter uses the latest version of the .NET run-time. For

managed applications like C# and Visual Basic .NET Operator Interfaces, the .NET Adapter uses the run time specified when the application was created.

You can load .NET 1.1 assemblies using the Microsoft Visual Studio 2005 run-time in TestStand. However, the .NET 1.1 run-time returns an error when you attempt to load .NET 2.0 assemblies. In addition, when you use Microsoft Visual Studio .NET 2003, attempts to debug the Common Run-time Language fail if the .NET 2.0 run-time is loaded in memory. TestStand also fails when you attempt to step into .NET assemblies. This is expected behavior, confirmed by Microsoft.

You can force an application to use a specific version of the .NET Framework by creating a configuration file in the same directory as the executable.

For example, to force the TestStand Sequence Editor to use .NET Framework 1.1, create a file called `SeqEdit.exe.config` with the following contents:

```
<?xml version="1.0"?>
<configuration>
    <runtime>
        <assemblyBinding
        xmlns="urn:schemas-microsoft-com:asm.v1">
        </assemblyBinding>
    </runtime>
    <startup>
        <supportedRuntime version="v1.1.4322" />
    </startup>
</configuration>
```

Debugging a Code Module within a TestStand Process

If you use Microsoft Visual Studio .NET 2003 to launch or attach to a TestStand process for debugging a code module called by the C/C++ DLL Adapter or the .NET Adapter, you must ensure that the Visual Studio project for the code module allows you to debug unmanaged code. If you do not do this, the Step Into command in TestStand will not work.

In Visual Studio, use the Project Property Pages dialog box to specify which debugger to use. Select **Project»Properties** in Visual Studio to launch the Project Property Pages dialog box. Since the options for enabling debuggers vary according to the type of project, use Table 2 as a guideline for each project type.

Table 2. Recommended Debugger Settings by Project Type

Project Type	Recommended Debugger Setting
Visual Basic .NET	Visual Basic projects automatically allow you to debug managed code. To allow TestStand to step into a code module, enable the **Unmanaged code debugging** option in the Configuration Properties group of the Project Property Pages dialog box.
C# .NET	C# projects automatically allow you to debug managed code. To allow TestStand to step into a code module, enable the **Enable Unmanaged Debugging** option in the Debugging subgroup in the Configuration Properties group of the Project Property Pages dialog box.
C++ .NET	C++ .NET projects can contain both managed and unmanaged code. The Debugger Type option in the Debugging subgroup in the Configuration Properties group of the Project Property Pages dialog box specifies one of the following options: • **Native Only**—Debugs unmanaged C++ code only. • **Managed Only**—Debugs managed code that runs under the common language run time. • **Mixed**—Invokes debuggers for both managed and unmanaged code. Use this setting to allow TestStand to step into a code module. • **Auto**—Attempts to determine the debugger type based on compiler and EXE information. This is the default setting. **Note**: In some instances, the Auto option incorrectly selects the managed code debugger instead of selecting both managed and unmanaged.
C++ (Native)	Native C++ projects, such as Win32 projects, already allow you to debug unmanaged code.

TestStand 3.5 Documentation

Note The TestStand 3.1 and 3.5 features and API additions have not been added to the printed manuals or posters. Refer to the *TestStand Help* for new feature documentation as specified in the following list. Refer to the *TestStand API Enhancements* section for a complete listing of the new properties and methods in TestStand 3.5.

The following documents have been updated for TestStand 3.5:

- *TestStand 3.5 Quick Start Guide*
- *TestStand Help*
  - *TestStand API Reference Help*—Additional documentation for the new API enhancements in TestStand 3.5.

The following electronic resources are available:

- *TestStand Bookshelf*—Use the *TestStand Bookshelf* to access all of the documentation in electronic format. You can also search the *TestStand Bookshelf* by keyword and/or phrase to find information quickly. To access the *TestStand Bookshelf*, select **Start»Programs»National Instruments»TestStand 3.5»Online Help»TestStand Bookshelf**.
- *TestStand Help*—Contains information about the TestStand environment and the TestStand UI Controls and Engine APIs. The *TestStand Help* also includes basic information about using an ActiveX automation server. To access the *TestStand Help*, select **Start»Programs»National Instruments»TestStand 3.5»Online Help»TestStand Help**.

Note If you are opening the *TestStand Help* from the `<TestStand>/Doc/Help` directory, National Instruments recommends that you open the `TSHelp.chm` file. The `TSHelp.chm` file is a collection of all of the help files available in TestStand and provides a complete table of contents and index.

TestStand 3.0 Documentation Updates

The following sections describe changes to the unrevised, printed TestStand documentation for TestStand 3.5. These changes will be incorporated into future revisions of the TestStand documentation set.

TestStand Reference Manual

- Change the *Step Execution* section on page 3-12 of Chapter 3, *Executions*, as follows:

  1. Change the order of actions in Table 3-4 to match the order of actions listed in the following table.

Table 3-4. Order of Actions that a Step Performs

Action Number	Description	Remarks
1	Allocate step result	—
2	Enter batch synchronization section	If option is set
3	Evaluate precondition	If `False`, perform Action Number 23, then proceed to Action Number 27
4	Acquire step lock	If option is set
⋮	⋮	⋮
27	Release step lock	If option is set
28	Exit batch synchronization section	If option is set
29	Fill out step result	—
30	Call PostResultListEntryEngine callback	—

  2. Change the last sentence in the paragraph after Table 3-4 to the following:

     If these callbacks are not defined or if they do not reset the error state for the step, TestStand performs Action Number 23 and then proceeds to Action Number 27. If a run-time error occurs in a loop iteration, TestStand performs Action Number 19 before performing Action Number 23 and 27.

- Replace the first paragraph of the *Creating Type Libraries* section on page 5-5 of Chapter 5, *Module Adapters*, with the following text:

  A type library exposes the function names and arguments of a DLL or COM object. Typically, type libraries are part of the DLL itself. TestStand does not require you to create a DLL with a built-in type library. However, if you include a type library in your DLL, the C/C++ DLL Adapter and LabWindows/CVI Adapter can use the information in the type library to obtain the parameter list information. If you use LabWindows/CVI 7.1 or later, TestStand reads function parameter information directly from the DLL without the use of a type library.

- Change the *Setting up TestStand as a Server for Remote Execution* section on page 5-15 of Chapter 5, *Module Adapters* as follows:

  1. Replace the tip on page 5-16 with the following:

Tip To minimize the configuration of security permissions, enable the **Allow All Users Access From Remote Machines** option on the Station Options dialog box. When you enable this option, TestStand configures the security permissions for you by adding the name Everyone to the list of users who have permission to launch the TestStand remote server. When you disable this option, TestStand removes the name Everyone from the list. In Windows XP Service Pack 2 or later, you will need to configure the permission limits for your machine as described in the following section.

  2. Change the following steps in the *Windows XP* section as follows:
     - Add the following note after Step 4.

Note If you did not enable the Allow All Users Access From Remote Machines option on the Station Options dialog box, the changes in Steps 5 through 7 are unnecessary and you should follow Steps 8 through 10 for making changes in the NI TestStand Remote Engine Properties dialog box.

     - Remove the following sentence from Step 5.

       Click **OK** to close the dialog box.
     - Add the following steps after Step 5. Renumber the remaining steps as needed.

       6. If you are using Windows XP Service Pack 2 or later, and you have enabled the Allow All Users Access From Remote Machines option on the Station Options dialog box, click the **COM Security** tab of the My Computer Properties dialog box.

          a. Click **Edit Limits** in the Access Permissions section to launch the Access Permission dialog box. Select **ANONYMOUS LOGON** and enable **Remote Access** in

the Permissions for ANONYMOUS LOGON section. Click **OK** to close the Access Permission dialog box.

b. Click **Edit Limits** in the Launch and Activation Permissions section to launch the Launch Permission dialog box. Select **Everyone** and enable **Remote Launch** and **Remote Activation** in the Permissions for Everyone section. Click **OK** to close the Launch Permission dialog box.

7. Click **OK** to close the My Computer Properties dialog box.

Note You do not need to change the settings on the NI TestStand Remote Engine Properties dialog box as described in Steps 8 through 10 if you have enabled the Allow All Users Access From Remote Machines option on the Station Options dialog box.

- Change the directory search order in the *Creating String Resource Files* section on page 8-6 of Chapter 8, *Customizing and Configuring TestStand*, to the following:

1. `<TestStand>\Components\User\Language\<current language>`
2. `<TestStand>\Components\User\Language\English`
3. `<TestStand>\Components\User\Language`
4. `<TestStand>\Components\NI\Language\<current language>`
5. `<TestStand>\Components\NI\Language\English`
6. `<TestStand>\Components\NI\Language`

- Add the following text at the end of the *Visual C++* section on page 9-14 of Chapter 9, *Creating Custom Operator Interfaces*.

Obtaining an Interface Pointer and CWnd for an ActiveX Control

You can use the following methods to obtain an interface pointer to an ActiveX control, such as a TestStand UI control, that you insert into an MFC dialog resource.

Method 1: Use GetDlgItem

1. Add a CWnd member to the dialog class for the control as follows:

```
CWnd mExprEditCWnd;
```

2. Insert the following code into the OnInitDialog method of the dialog class:

```
mExprEditCWnd.Attach(GetDlgItem
(IDC_MYEXPRESSIONEDIT)->m_hWnd);
```

3. Obtain the interface pointer from the CWnd member as follows:

```
TSUI::IExpressionEditPtr myExprEdit =
mExprEditCWnd.GetControlUnknown();
```

Note If you are using MFC 7.0 or later, you cannot call GetDlgItem for windowless ActiveX controls. The following TestStand UI controls are windowless and must use Method 2: Application Manager control, SequenceFileView Manager control, and ExecutionView Manager control.

Method 2: Use DoDataExchange

1. Add a CWnd member to the dialog class for the control as follows:

```
CWnd mMySequenceFileViewMgrCWnd;
```

2. Add an entry to the AFX_DATA_MAP in your DoDataExchange method to initialize the CWnd as follows:

```
DDX_Control(pDX, IDC_MY_SEQUENCE_FILE_VIEW_MGR,
mMySequenceFileViewMgrCWnd);
```

3. Obtain the interface pointer from the CWnd member as follows:

```
TSUI::ISequenceFileViewMgrPtr mySequenceFileViewMgr
= mMySequenceFileViewMgrCWnd.GetControlUnknown();
```

Note Typically, ActiveX controls do not document whether they recreate their window. Only use Method 2 for ActiveX controls that are windowless or do not recreate their internal window.

- Replace the first note on page 10-10 in the *Engine Callbacks* section of Chapter 10, *Customizing Process Models and Callbacks* with the following text:

Note If a callback sequence is empty, TestStand does not invoke the Engine callback. Also, the process model uses the Execution.EnableCallback method to disable the ProcessModelPostResultListEntry and SequenceFilePostResultListEntry callbacks when the model does not need to process results on-the-fly to generate a report or log to database.

- Replace the *General Tab* section on page 13-4 of Chapter 13, *Creating Custom Step Types* with the following text:

Use the General tab to specify the icon, description, and comment for the step type. You can specify the default name for new steps of this type. You can also designate a single module adapter for steps of this type and configure the default module that steps call.

Note If you want a step type to always call a specific code module and you do not want the sequence developer to configure the module call, define a Post-Step substep on the Substeps tab instead of specifying a default module.

- Change the first sentence in the third paragraph of the *File Collection* section on page 14-3 in Chapter 14, *Deploying TestStand Systems* to the following:

  The `filter.ini` file, located in the `<TestStand>\Cfg` directory, defines those files that the deployment utility automatically excludes from any deployment package it creates.

- Add the following section after the *Sequence File Processing* section on page 14-5 of Chapter 14, *Deploying TestStand Systems*.

Installing National Instruments Components

The TestStand Deployment Utility allows you to package National Instruments hardware drivers and other components, such as run-time engines, in deployment installers. You can configure the additional components included in the deployment using the Drivers and Components dialog box.

To launch the Drivers and Components dialog box, launch the Deploy TestStand System dialog box and click **Drivers and Components** on the **Installer Options** tab. Refer to the *TestStand Help* for more information about the Deploy TestStand System and Drivers and Components dialog boxes.

The Drivers and Components dialog box only lists components that are installed on your computer and were installed from the NI Device Driver CD that ships with TestStand 3.5, or a later version of the driver CD. The components you select contain only the features currently installed on your computer, and therefore, may not be complete copies of the original product(s).

- Add the following note after the first bullet in the *Guidelines for Successful Deployment* section on page 14-5 in Chapter 14, *Deploying TestStand Systems*.

Note If you are using LabVIEW 8.0 or later, you cannot create deployments that contain duplicate VIs or subcomponents, such as DLLs. You must ensure that all sequences included in the deployment image reference the same VI and DLL files.

- Add the following bullet to the end of the bulleted list in the *Guidelines for Successful Deployment* section on page 14-6 of Chapter 14, *Deploying TestStand Systems*.

  - **Redeploy edited files**—If you edit any of your deployed system files, the deployed system may no longer function and you must redeploy the system.

- Replace step 5 in the *Distributing an Operator Interface* section on page 14-10 of Chapter 14, *Deploying TestStand Systems* with the following text:

    5. Select both `TestExec.exe` and `TestExec.uir` and click **Add**.

- Add the following note at the beginning of Appendix C, *IVI Step Types*.

Note The IVI step types support the IVI-C class compliant instrument drivers. You can use the ActiveX/COM Adapter when you use the IVI-COM class compliant instrument drivers.

- Add the following note at the end of the *Loading from File* section on page D-6 of Appendix D, *Database Step Types*.

Note When you specify starting and ending data markers in the Import/Export Properties dialog box, enter the marker text in the text controls without double quotes. However, when you specify starting and ending data markers in the expression controls of the Edit Property Loader dialog box, you must surround literal marker text values with double quotes.

Using TestStand

- Replace the last bullet on page 3-1 of Chapter 3, *Editing Steps in a Sequence* with the following:

    - IVI-C (Dmm, Scope, Fgen, Power Supply, Switch, Tools)

Using LabVIEW with TestStand Manual

- Change the TestStand Data Type information for ActiveX Control, Automation Refnum, or .NET Refnum in Table 4-1 in the *Data Type Conversion* section on page 4-1 of Chapter 4, *Using LabVIEW Data Types with TestStand* to the following:

    You can also store references to .NET objects that you create in LabVIEW within TestStand properties and then pass them to other LabVIEW VIs.

    When you create a .NET object with LabVIEW 7.*x* or earlier, the object must be *marshallable by ref*.

- Add the following text after the third paragraph in the *Format of Legacy VIs* section on page C-1 of Appendix C, *Calling Legacy VIs*.

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

    - Using the **Test Data** cluster.

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

However, if you use both methods simultaneously, the values set using the sequence context ActiveX reference take precedence over the values set using the **Test Data** cluster. In other words, if you use both methods to set the value of the same variable, the values you set using the sequence context ActiveX reference are recognized. The values you set using the **Test Data** cluster are ignored.

Note You can use both the sequence context ActiveX reference and the **Test Data** cluster together in your code module provided that you do not try to set the same variable twice. For example, if you use the sequence context ActiveX reference to set the value of Step.Result.PassFail and then use the **Test Data** cluster to set the value of Step.Result.ReportText, both values are set correctly.

Using LabWindows/CVI with TestStand Manual

- Add the following note after Table 4-1 in the *Data Type Conversion* section on page 4-1 of Chapter 4, *Using LabWindows/CVI Data Types with TestStand*.

Note The LabWindows/CVI Adapter supports return values of type numeric, which includes Boolean, and void.

- The following type definition should replace the first type definition in Step 19 on page 4-6 in Chapter 4, *Using LabWindows/CVI Data Types with TestStand*.

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

- Add the following text at the end of the *Adding and Releasing References* section on page B-5 of Appendix B, *Using the TestStand ActiveX APIs in LabWindows/CVI*.

While many of the functions specified in the tsapicvi library are simple wrappers to API methods that require no storage of information, there are several functions, especially those containing Get or New, where TestStand is actively allocating new memory to hold the information being passed to LabWindows/CVI. In any instance where you are using a function of this type, you must release the allocated memory at the end of your code using calls to CA_FreeMemory, CA_DiscardObjHandle, or a similar function.

If you are concerned about whether a function returns a piece of data that needs to be manually released, refer to the *LabWindows/CVI Help* or the *TestStand Help* for that function. Both of these resources explicitly state if the function is allocating memory and often contain additional code fragments explaining how to use the function.

The following are examples of functions that allocate memory:

```
TS_PropertyGetValString()
TS_PropertyGetValIDispatch()
TS_PropertyGetPropertyObject()
TS_NewEngine()
TS_SeqFileNewEditContext()
TS_EngineNewSequence()
```

The following is an example of using one of these functions and then releasing the memory:

```
char *stringVal = NULL;
TS_PropertyGetValString (propObj, &errorInfo,
    "Step.Limits.String", 0, &stringVal);
CA_FreeMemory (stringVal);
```

- Change the *Automatically Updated Step Properties* section on page C-4 of Appendix C, *Calling Legacy Code Modules* as follows:

1. Rename the section *Updating Step Properties*.
2. Add the following text to the beginning of the section:

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

   - Using the **tTestData** structure.
   - Using the sequence context ActiveX reference. This method allows you to call the TestStand ActiveX API functions to set the variables used to store the results of your test, such as Step.Result.PassFail.
3. Add the following note after Table C-3:

Note The values set using the sequence context ActiveX reference take precedence over the values set using the **tTestData** structure. In other words, if you use both methods to set the value of the same variable, the values you set using the sequence context ActiveX reference are recognized. The values you set using the **tTestData** structure are ignored.

You can use both the sequence context ActiveX reference and the **tTestData** structure together in your code module provided that you do not try to set the same variable twice. For example, if you use the sequence context ActiveX reference to set the value of Step.Result.PassFail and then use the **tTestData** structure to set the value of Step.Result.ReportText, both values are set correctly.

TestStand System and Architecture Overview Card

- Change the IVI Steps to IVI-C Steps in the Built-In Step Types section of the Architecture Overview diagram.
- Remove `.BAS` from the Code Modules section of the Architecture Overview diagram. TestStand does not support the `.BAS` file type.

National Instruments™, NI™, ni.com™, and LabVIEW™ are trademarks of National Instruments. Product and company names listed are trademarks or trade names of their respective companies. For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

© 1999–2005 National Instruments Corporation. All rights reserved.

372519K-01 Oct05