RELEASE NOTES

TestStand™

Version 3.1

These release notes contain system requirements for TestStand 3.1, as well as information about new features, documentation resources, and other changes since TestStand 3.0.

Contents

NATIONAL INSTRUMENTS™

Minimum System Requirements

To run TestStand 3.1, National Instruments recommends that your system meet the following requirements:

- Windows 2000/XP or Windows NT 4.0 Service Pack 6a or later required
- 600 MHz Pentium class microprocessor (400 MHz minimum)
- 128 MB of memory (256 MB recommended)
- 500 MB of free hard disk space (100 MB minimum)
- 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 3.1 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.
- 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 3.1 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**.

   The TestStand Uninstaller launches 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 3.1 into this directory.

Although TestStand 3.1 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 3.1 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 3.1»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 3.1 and run an operator interface from your previous TestStand installation, the operator interface uses the engine, step types, and components from TestStand 3.1. If you activate your previous installation of TestStand and run a TestStand 3.1 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 3.1 installation directory:

- <TestStand>\Components\User
- <TestStand>\CodeTemplates\User
- <TestStand>\OperatorInterfaces\User
- <TestStand>\Cfg\TypePalettes\MyTypes.ini
- <TestStand>\Cfg\TypePalettes\SeqEdit.ini

- `<TestStand>\Cfg\TypePalettes\TestExec.ini`
- `<TestStand>\Cfg\TypePalettes\TestStandDatabaseOptions.ini`
- `<TestStand>\Cfg\TypePalettes\TestStandModelModelOptions.ini`
- `<TestStand>\Cfg\TypePalettes\TestStandModelReportOptions.ini`
- `<TestStand>\Cfg\TypePalettes\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 3.1:

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 3.1:

   `<TestStand>\Setup\ToolMenusToInstall`.
4. Place the Tools menu file in the new directory.
5. Launch the TestStand 3.1 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 the 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 3.1. 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 3.1, 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 3.1 version of the component and the original version of the component that you modified. Then, apply the TestStand 3.1 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 3.1 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 3.1 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 are located 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 3.0 and version 3.1.

- In TestStand 3.0, the ListBar control fired the CurPageChanged event when you selected a new current page or when you changed a property of the current page. In TestStand 3.1, the ListBar control only fires the CurPageChanged event when you select a new current page.

- Typically, TestStand does not call Custom substeps. In TestStand 3.1, if you add a Custom substep named OnNewStep to a step type, the TestStand Sequence Editor calls the substep each time you create a new step of that type. You can use the TestStand API to invoke a Custom substep from a code module or an operator interface.
- In TestStand 3.1, the PropertyObject.GetXML method generates a stream that contains additional XML attributes and elements to support the new PropertyObject.SetXML method.

  Additionally, the values of the Type and ElementType attributes no longer specify named types. The new TypeName and ElementTypeName attributes specify named types, and the Type and ElementType attributes specify base types only.

  Refer to the following file, `<TestStand>\Components\Models\TestStandModels\PropertyObject.xsd`, which defines the XSD schema definition for the PropertyObject.GetXML and PropertyObject.SetXML methods.

What's New in TestStand 3.1

This section describes the new features in TestStand 3.1 and other changes since TestStand 3.0.

Flow Control Step Types

TestStand 3.1 introduces the Flow Control step types. You can use the following new step types to implement common programming constructs:

- **If**—Defines a block of steps that execute if a condition is met.
- **Else**—Defines a block of steps that execute if the condition defined by the preceding If or Else If step is not met.
- **Else If**—Defines a block of steps that execute if a condition is met and the conditions defined by the preceding If step and any preceding Else If steps are not met.
- **For**—Defines a block of steps that execute repeatedly for a number of iterations.
- **For Each**—Defines a block of steps that execute once for each element in an array.
- **While**—Defines a block of steps that execute repeatedly while a condition is `True`.
- **Do While**—Defines a block of steps that execute once and then repeatedly while a condition is `True`.
- **Break**—Causes a For, For Each, While, or Do While loop block or a Case block to exit before completing.

- **Continue**—Causes the next iteration of an enclosing For, For Each, While, or Do While loop block to begin.
- **Select**—Defines a block of steps that encloses sub-blocks defined by Case steps. The Select step specifies an expression that determines which Case block executes.
- **Case**—Defines a block of steps within a Select block that execute if the expression specified by the Select step evaluates to a certain value.
- **End**—Defines the end of any block of steps.

Refer to the *TestStand Help* for more information about the dialog boxes associated with the Flow Control step types.

Synchronization Step Types

TestStand 3.1 introduces new step types for the automatic scheduling of steps that require access to resources (for example, hardware resources). The new step types are as follows:

- **Auto Schedule**—Defines a block that contains any number of Use Auto Scheduled Resource sub-blocks. The Auto Schedule step executes each sub-block once. The order in which the step executes the sub-blocks can vary according to the availability of the resources that the sub-blocks require. Typically, you use Auto Schedule steps in a sequence that you execute using the Parallel or Batch process model. The Auto Schedule block can increase CPU and/or resource utilization by directing a thread that would otherwise wait for a resource locked by another thread to perform other actions using an available resource instead.
- **Use Auto Scheduled Resource**—Defines a sub-block of steps within an Auto Schedule block that use a specified resource or set of resources. The Use Auto Scheduled Resource step locks the specified resources while the steps in its sub-block execute.

Refer to the *TestStand Help* for more information about the dialog boxes associated with the new Synchronization step types.

Refer to the `AutoSchedule.seq` file, located in `<TestStand>\Examples\Auto Schedule`, for an example of how to implement auto scheduling in a sequence. This example demonstrates how the Auto Schedule step type can improve performance when testing multiple UUTs in parallel using shared hardware resources. You can use the Auto Schedule step to dynamically determine the order in which a UUT uses its required resources in order to minimize delays that occur while the UUT waits for a resource to become available.

LabVIEW Express VI Instantiation

TestStand 3.1 adds the ability to directly specify and configure an Express VI to call from a TestStand sequence.

The Edit LabVIEW VI Call dialog box contains a new combo box in which you can select which type of VI to call. If you select Express VI from the combo box, the Browse button displays the LabVIEW icon instead of the open file icon. Click the LabVIEW icon to open a context menu which displays all of the Express VIs in the currently selected LabVIEW function palette.

Note You can only configure calls to Express VIs when the LabVIEW Adapter is configured to use the LabVIEW Development System. If you configure the LabVIEW Adapter to use the LabVIEW Run-Time Engine, you will not be able to select and configure an Express VI. Additionally, your active LabVIEW Development System must be version 7.1 or later to use Express VIs.

Enhanced LabVIEW and LabVIEW Real-Time Integration

LabVIEW Utility Step Types

TestStand 3.1 introduces new step types for communicating with LabVIEW remote systems. The new step types are as follows:

- **Check Remote System Status**—Determines whether a remote system is running LabVIEW and allows TestStand to connect to it. This step type is particularly useful for checking the status of LabVIEW Real-Time systems.
- **Run VI Asynchronously**—Runs a VI in a new thread in the TestStand execution. This step type is particularly useful for downloading and running a VI asynchronously on a LabVIEW Real-Time system.

Refer to the *TestStand Help* for more information about the dialog boxes associated with the LabVIEW Utility step types.

LabVIEW Real-Time Error Messaging

TestStand 3.1 also improves the error messaging for LabVIEW Real-Time. The following error cases now contain more descriptive error messages:

- Unknown hostname
- Host unreachable
- Host not allowing TCP/IP connection
- Host not allowing VI Server connection
- Host not allowing VI download
- VI broken

- VI not found
- Different LabVIEW version when doing LabVIEW Real-Time download
- Full-featured LabVIEW Run-Time Engine not present

FTP Files Step Type

TestStand 3.1 introduces the new FTP Files built-in step type. This step type gives you the ability to download files to and upload files from remote systems. This step type is particularly useful if you are using a LabVIEW Real-Time system.

CheckBox User Interface (UI) Control

TestStand 3.1 introduces the CheckBox control. Connect an Application Manager, SequenceFileView Manager, or ExecutionView Manager control to the CheckBox control to connect to commands that you can toggle.

Refer to the *TestStand Help* for detailed descriptions of the properties, methods, and events associated with the CheckBox control. Refer to Table 2 for a general listing of the new API.

Property Loader

TestStand 3.1 includes the following improvements to the Property Loader:

- The Property Loader step type has the ability to import properties into any target sequence, not only the current sequence. The imported values are then used for subsequent calls to that sequence in the same execution.
- The Property Loader step type and Import/Export Properties dialog box have the ability to import and export array and container values as XML strings.

ExpressionEdit Control Enhancements

In TestStand 3.1, the ExpressionEdit control contains the following new features:

- Addition of Browse Expression and Check Expression buttons to the ExpressionEdit control
- Ability to add custom buttons
- On-the-fly syntax checking
- Variable creation
- Ability to change the ExpressionEdit control to a combo box

User-Defined Location for Configuration Files

TestStand 3.1 introduces the ability to modify the location of the TestStand configuration file directory. By default, the directory is `<TestStand>\Cfg`. This new capability allows users who do not have write access to the TestStand installation directory to run TestStand. You can specify the configuration file directory on the Preferences tab of the Station Options dialog box.

Configuration Files and Type Palettes

In TestStand versions 2.0 through 3.0, TestStand installed type palettes in the `Cfg` directory. Also, in TestStand 3.0, the TestStand 2.0 and 2.0.1 type palette files were installed in the `Cfg\Compatibility` directory, which is used for the Save As feature. In TestStand 3.1, both directories are now located in the `Components` directory.

Note You should not use the `Cfg` or `Cfg\Compatibility` directories for saving your type palette files.

If you create a new type palette, the type palette location defaults to the `User\TypePalette` directory. If you are using the Save As feature in TestStand, save your type palette files in the following directory: `Components\User\Compatibility\Version`.

Configuration Files and the TestStand Deployment Utility

The TestStand Deployment Utility also allows users to create an installer that sets a non-default location from the configuration file directory.

To specify the configuration files directory, click the **Advanced Options** button on the Installer Options tab of the Deploy TestStand System dialog box to launch the Advanced Installer Options dialog box.

Refer to the *TestStand Help* for more information about the Deploy TestStand System dialog box and the Advanced Installer Options dialog box.

Running Executables from Deployment Installers

The TestStand Deployment Utility allows you to create an installer in which you can run a sequence of user-specified commands. For example, you can specify an executable and its command-line arguments.

To specify the commands, click the **Custom Commands** button on the Installer Options tab of the Deploy TestStand System dialog box to launch

the Custom Commands dialog box. You can specify the executable, the command-line arguments, and the order of execution in this dialog box. The dialog box also includes a set of macros for entering special directories into the command-line arguments.

You can also display a dialog box which shows what command is currently executing and request a reboot after all commands are finished.

Refer to the *TestStand Help* for more information about the Custom Commands dialog box.

Database Logging Enhancements

TestStand 3.1 includes the following improvements to the process model database logging functionality:

- Supports ODBC drivers that require string data to be passed as a variant with VARCHAR data rather than BSTR data.
- Supports user-specified expressions that generate primary key values.
- Includes new default schemas and SQL script files to support MySQL databases and Sybase Adaptive Anywhere databases.

TestStand API Enhancements

The following tables list the new properties, methods, and events that have been added to the TestStand API and the TestStand User Interface (UI) Controls API in TestStand 3.1.

Table 1. New TestStand API Properties and Methods

Class	Properties	Methods
ActiveXParameter	UserData (Read Only)	—
CommonCParameter	UserData (Read Only)	—
DotNetParameter	ArrayDimensions (Read Only) UserData (Read Only)	—
Engine	—	GetLocationForNextDialog SetConfigDirectory
Execution	—	ClearSequenceDefaultValues GetSequenceDefaultValues NewSequenceDefaultValues SetSequenceDefaultValues
LabVIEWModule	ExpressVIName (Read Only) VIAttached (Read Only) VIType	ExportVI ImportVI
LabVIEWParameter	ArrayDimensions (Read Only) UserData (Read Only)	GetArrayIndex GetDefaultArrayDimensionSize

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

Class	Properties	Methods
LabVIEWParameterElement	ArrayDimensions (Read Only) UserData (Read Only)	GetArrayIndex GetDefaultArrayDimensionSize
PropertyObject	—	GetValBinary SetValBinary SetXML
PropertyObjectFile	Requirements (Read Only)	—
Sequence	HasMismatchedBlocks (Read Only) Requirements (Read Only)	—
SequenceCallModule	—	LoadParametersFromSequence
SequenceCallParameter	UserData (Read Only)	—
SequenceContext	NumStepsExecuted	—
SequenceFile	—	AddLoadReference
StationOptions	BreakOnSequenceFailure BreakOnStepFailure	—
Step	BlockEndIndex BlockFlags BlockLevel BlockLevelsUnmatched BlockNextIndex BlockParentIndex BlockPreviousIndex BlockStartIndex Requirements (Read Only)	—
StepType	AppliesToBlockStructure BlockEndTypes BlockStartTypes	—

Table 2. New TestStand User Interface Controls API Properties, Methods, and Events

Class	Properties	Methods	Events
ApplicationMgr	ApplicationWillExitOnStart (Read Only) ConfigurationEntryPoints (Read Only) ExecutionEntryPoints (Read Only)	OpenSequenceFilesDialog	AfterUIMessageEvent ProcessUserCommands QueryReloadSequenceFile
CheckBox	BackColor Caption Checked Enabled Font FontSource ForeColor hWnd (Read Only) Image ImageAlign MaskColor ScaleWithDPI Style TextAlign UseMaskColor	Localize	Click KeyDown KeyPress KeyUp MouseDown MouseMove MouseUp
ExpressionEdit	Buttons (Read Only) ComboBoxItems (Read Only) DisplayText DropDownListHwnd (Read Only) ShowDisplayNameWhenInactive Style	Localize	ButtonClick DropDown InsertComboBoxItem
ExpressionEditButton	BackColor ContextMenuItemCaption Enabled hWnd (Read Only) Icon Kind ShortcutKey ShortcutModifier ToolTipText Visible	—	—
ExpressionEditButtons	Count (Read Only)	Clear GetItem Insert Remove	—
ExpressionEditComboBoxItem	DisplayName Icon Value	—	—
ExpressionEditComboBoxItems	Count (Read Only) Item (Read Only)	Clear Insert Remove	—
SequenceView	BlockDisplayOptions	—	—

Refer to the *TestStand Help* for more information about the classes, properties, methods, and events listed in Tables 1 and 2.

Miscellaneous Features

- ATLAS step type example.
- Automated Test Markup Language (ATML) Test Results Markup Language (TRML) result logging example.
- LabWindows/CVI 7.1 DLLs export function and parameter information without a type library. The TestStand 3.1 C/C++ DLL Adapter and LabWindows/CVI Adapter can read this information.

Using Microsoft Visual Studio .NET to Debug a Code Module within a TestStand Process

If you use Microsoft Visual Studio .NET 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 from TestStand will not work.

In Visual Studio, use the Project Properties dialog box to specify which debugger to use. Select **Project»Properties** in Visual Studio to launch the Project Properties dialog box. Since the options for enabling debuggers vary according to the type of project, use Table 3 as a guideline for each project type.

Table 3. 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 Properties 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 Configuration Properties group of the Project Properties dialog box.

Table 3. Recommended Debugger Settings by Project Type (Continued)

Project Type	Recommended Debugger Setting
C++ .NET	C++ .NET projects can contain both managed and unmanaged code. The Debugger Type option in the Configuration Properties group of the Project Properties dialog box specifies one of the following options: • **Native**—Debugs unmanaged C++ code only. • **Managed Only**—Debugs managed code that runs under the common language runtime. • **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 3.1 Documentation

Note The TestStand 3.1 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, methods, and events in TestStand 3.1.

The following documents have been updated for TestStand 3.1:

- *TestStand 3.1 Quick Start Guide*
- *TestStand Help*
  - *TestStand Environment Reference Help*—Documentation for the Flow Control, LabVIEW Utility, and additional Synchronization step types.
  - *TestStand UI Controls API Reference Help* and *TestStand API Reference Help*—Additional documentation for the new API enhancements in TestStand 3.1.
  - *TestStand Supplemental Help*—Addition of TestStand Synchronization Manager documentation.

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 3.1»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 3.1»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 3.1. 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

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

Action Number	Description	Remarks
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.

- Add the following bullet to the end of the bulleted list in the *Guidelines for Successful Deployment* section on page 14-5 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.
- 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 LabVIEW with TestStand Manual

- 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 the 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.

- 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.

CVI™, LabVIEW™, Measurement Studio™, National Instruments™, NI™, ni.com™, and TestStand™ are trademarks of National Instruments Corporation. Product and company names mentioned herein 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`.

© 1999–2004 National Instruments Corp. All rights reserved. Printed in Ireland.

322519G-01 Jul04