TestStand™ I: Introduction Course Manual

Course Software Version 3.5
November 2005 Edition
Part Number 322317H-01

Worldwide Technical Support and Product Information
ni.com

National Instruments Corporate Headquarters
11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 0 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 1800 226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 02 2377 2222, Thailand 662 278 6777, United Kingdom 44 0 1635 523545

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

Contents

Student Guide

Lesson 1
Introduction to TestStand

Lesson 2
TestStand Environment

Lesson 3
Creating Sequences

Lesson 4
TestStand Parameters, Variables, and Expressions

Lesson 5
Creating Code Modules in External Environments

Lesson 6
Importing and Exporting Properties

Lesson 7
Configuring TestStand

Lesson 8
User Management

Lesson 9
TestStand Types

Lesson 10
Database Interaction

Lesson 11
Custom Database Logging

Lesson 12
Distribution

Appendix A
Introduction to IVI

Appendix B
Additional Information and Resources

Course Evaluation

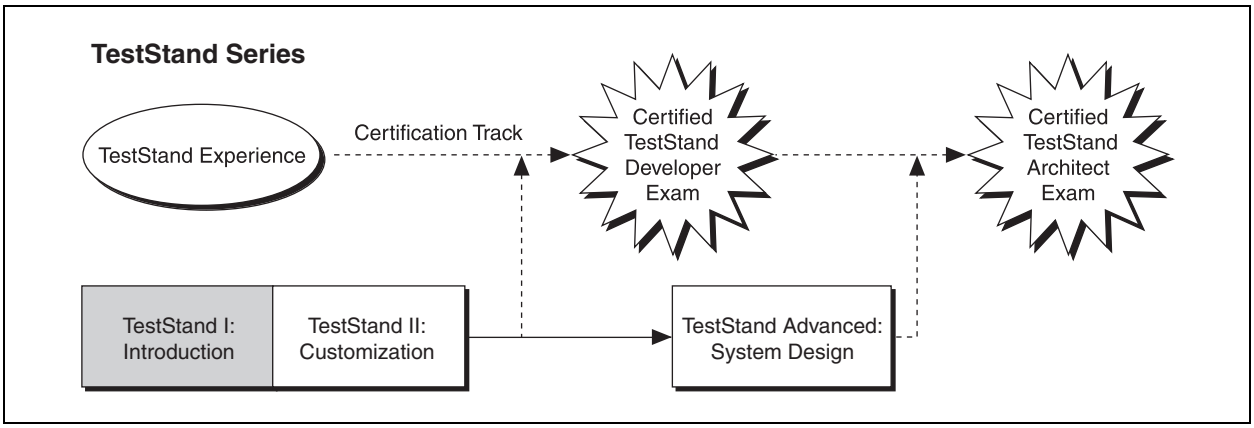
Student Guide

Thank you for purchasing the *TestStand I: Introduction* course kit. This course manual and the accompanying software are used in the three-day, hands-on *TestStand I: Introduction* course.

You can apply the full purchase of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit `ni.com/training` for online course schedules, syllabi, training centers, and class registration.

Note For course manual updates and corrections, refer to `ni.com/info` and enter the info code `rdts02`.

The *TestStand I: Introduction* course is part of a series of courses designed to build your proficiency with NI TestStand and help you prepare for exams to become an NI Certified TestStand Developer and NI Certified TestStand Architect. The following illustration shows the courses that are part of the TestStand training series. Refer to `ni.com/training` for more information about NI Certification.



A. About This Manual

This course introduces the TestStand environment and teaches you how to use TestStand for building automated test applications. The main emphasis of this course is to cover the fundamental features that TestStand offers. This course assumes you have a background knowledge of test executive software and a familiarity with LabVIEW or C programming.

The course is divided into lessons that teach a topic or a set of topics. Each lesson consists of the following elements:

- An introduction that describes the lesson objectives and the topics discussed.

- A discussion of the topics.
- A set of hands-on exercises that reinforce the concepts presented in the discussion. Some of the exercises are duplicated for LabVIEW and LabWindows™/CVI™ programming environments. For duplicate exercises, select the exercise that uses your preferred programming environment.
- A summary that highlights the concepts covered in the lesson.
- A self-review that allows you to test your mastery of the concepts covered in the lesson.

Note You can customize many of the topics discussed in this course to integrate further with your testing requirements.

B. What You Need to Get Started

Before you use this course manual, ensure you have all the following items:

- ❑ Computer running Windows 2000/NT/XP or later
- ❑ TestStand 3.5 or later
- ❑ LabVIEW 7.0 or later or LabWindows/CVI 7.0 or later
- ❑ *TestStand I: Introduction Course Manual*
- ❑ *TestStand I: Introduction* Course CD containing the following files:

Filename	Description
`Exercises`	Folder for saving files created during the course and files necessary to complete some exercises
`Solutions`	Folder containing the solutions to all the course exercises

Installing the Course Software

Install the course exercises and solutions by inserting the course CD and extracting the contents of the self-extracting archives into the `C:\Exercises\TestStand I` directory. The solutions to all exercises should be installed in the `C:\Solutions\TestStand I` directory.

The exercises in this course reinforce concepts presented in the lessons and provide additional explanations of the TestStand environment. Save your work in the `C:\Exercises\TestStand I` directory as you complete the exercises.

C. Course Goals

This TestStand course teaches you the fundamentals of TestStand. It describes the TestStand architecture, including an overview of each constituent part. The course also discusses the TestStand environment and demonstrates how to use TestStand to create a test management system. The course concludes by showing you how to distribute TestStand and your test sequences.

This course prepares you to do the following:

- Develop practical test applications in the TestStand environment and distribute them to test stations.
- Build test code and interface it to TestStand.
- Use TestStand debugging tools.
- Configure TestStand using the configuration dialog boxes and customize user privileges.
- Share data between TestStand and LabVIEW or LabWindows/CVI.
- Understand how a test executive functions
- Log test results to a database and use the Database Viewer to view logged data.

D. Course Conventions

The following conventions are used in this course manual:

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

This icon denotes a tip, which alerts you to advisory information.

This icon denotes a note, which alerts you to important information.

bold Bold text denotes items that you must select or click in the software, such as menu items and dialog box options, and text or characters that you should enter from the keyboard. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives,

	paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
monospace bold	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
monospace italic	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

1

Introduction to TestStand

Lesson 1: Introduction to TestStand

In this lesson, you will learn about:

- **Role of test management software**
- **TestStand architecture**
- **Essential TestStand concepts**

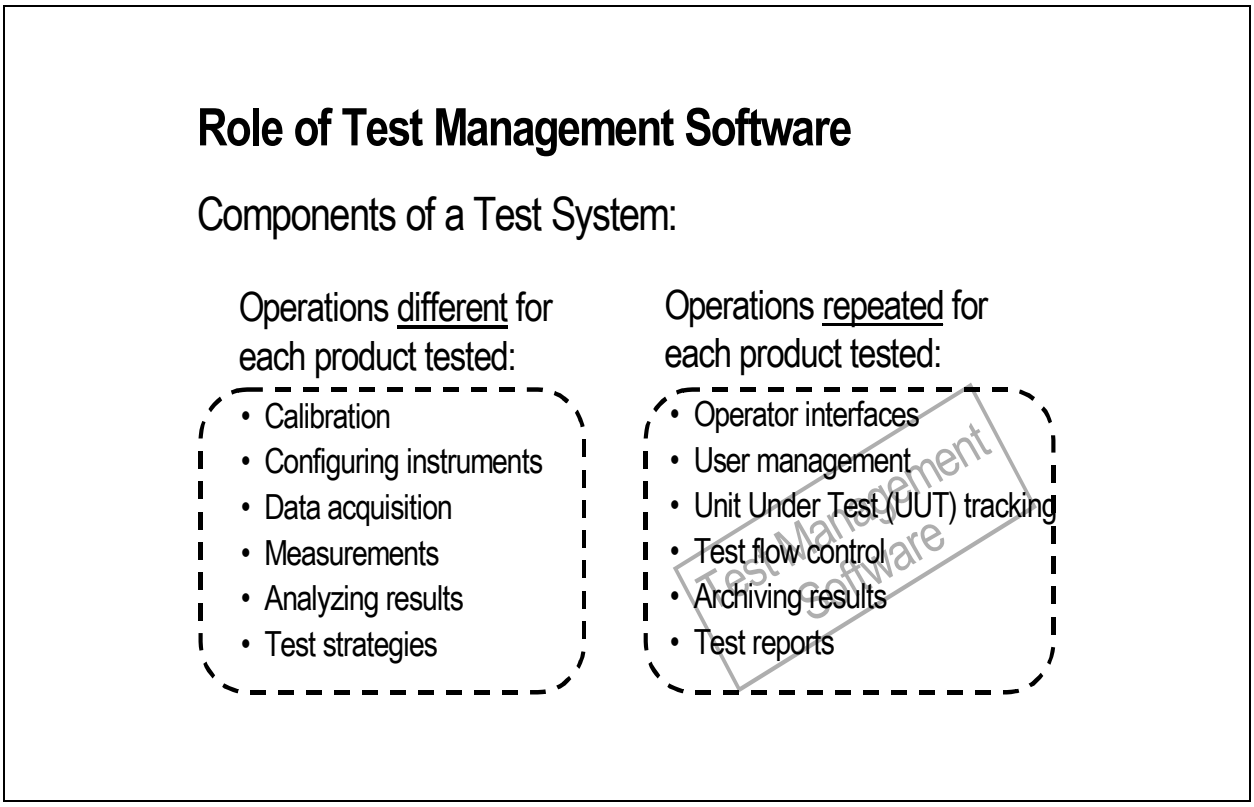
Introduction to TestStand

This lesson introduces the basic concepts of test management software and the components of the TestStand architecture.

Lesson one includes the following parts:

- Part one—Introduces the role of test management software and the tasks it performs.
- Part two—Addresses the question “What is TestStand?”
- Part three—Describes the TestStand architecture and provides detailed information about each component.

The exercises in this lesson introduce two main components of TestStand, the sequence editor and the operator interface.



Role of Test Management Software

In every testing environment, a wide variety of operations must be performed. These operations can be separated into two distinct groups: operations repeated for each product tested and operations different for each product tested.

Test management software provides a modular test framework to handle the operations that are repeated for each product. Operations such as displaying the user interface, providing user management, performing step branching, archiving results, and creating test reports are similar for each product tested, and can therefore be addressed through off-the-shelf test management software.

Operations that are different for each product tested, such as calibration and configuration routines, data acquisition and measurement, analysis of results, and developing test strategies are written in other development environments, such as LabVIEW and/or LabWindows/CVI. You can then implement these operations as steps in a test management application. Thus, test management software allows you to concentrate on developing tests for your products while reducing development time on the basic operations needed for testing environments. This allows you to develop test modules efficiently without redundant code development.

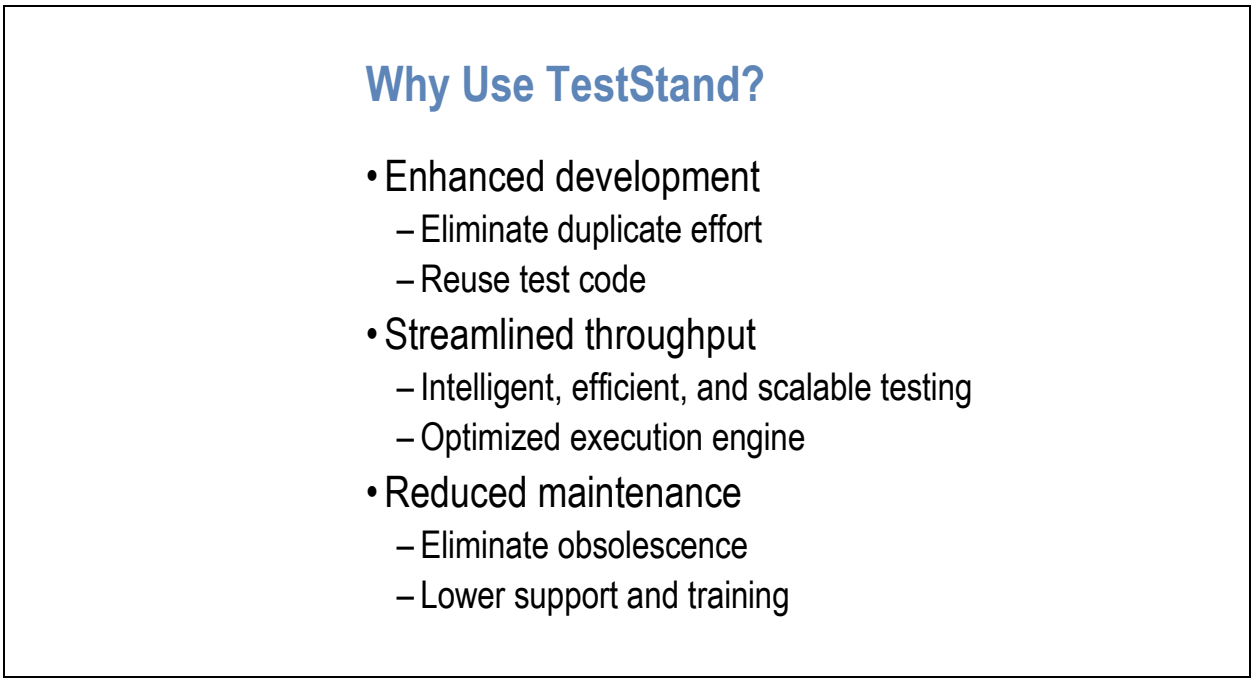
What is TestStand?

- Call tests written in any test language
- Parallel testing
- Automatic report generation
- Modular and customizable
- Flexible operator interfaces

What is TestStand?

National Instruments TestStand software, awarded the **2002 Test Product of the Year** by readers of *Test & Measurement World* magazine, is customizable, off-the-shelf test management software. TestStand is a full-featured test framework that provides flexibility to meet a diverse set of requirements. TestStand includes features such as parallel sequence execution, operator interfaces, report generation, database logging, and much more. TestStand gains its power and flexibility through the ability to customize many of these features.

NI TestStand is a flexible, open test management framework for building, customizing, and deploying a full-featured test management system. You can develop test sequences faster with TestStand by calling test programs created in the programming language of your choice. TestStand also is completely customizable, so you can modify and enhance it to match your specific needs. Using TestStand, you can focus your engineering efforts on more important initiatives while TestStand manages common test system tasks.

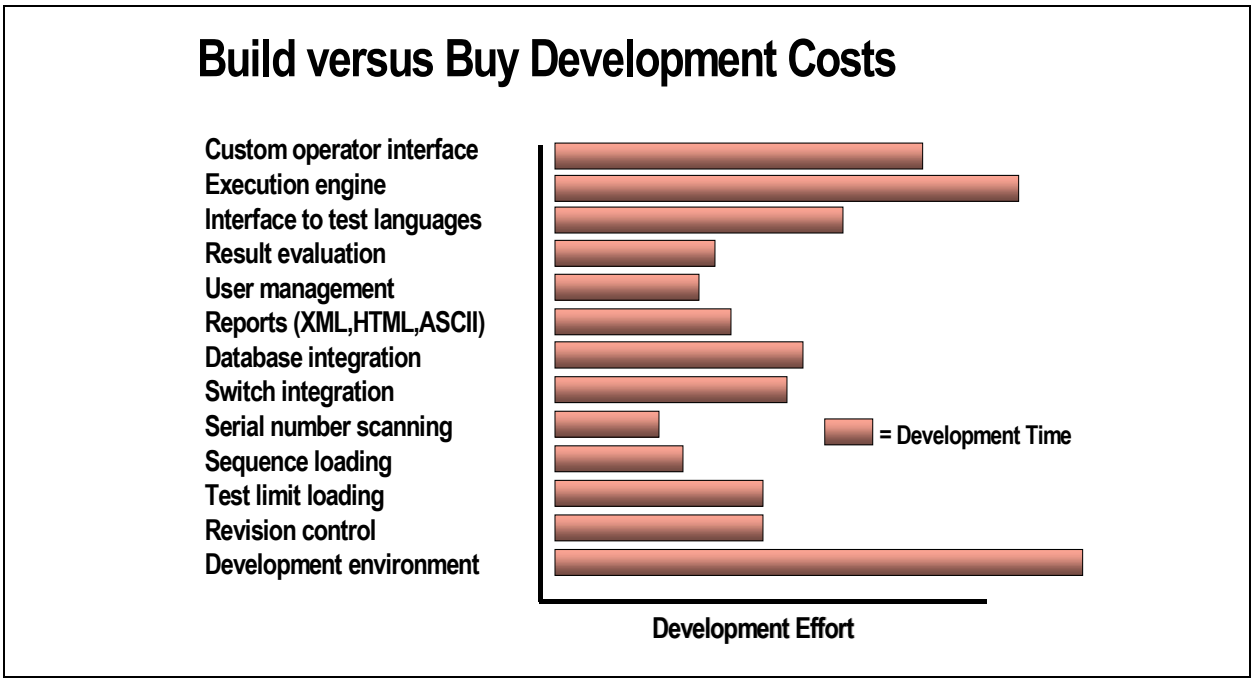


Why Use TestStand?

TestStand significantly reduces the amount of development time required to build automated test systems. By using TestStand, you immediately reduce development time because the core framework is provided. A modular test management framework is the most difficult part to develop in any test system, because it contains all the engine components, the decision making logic, and the reporting and database tools. Avoiding duplicated efforts saves years of development.

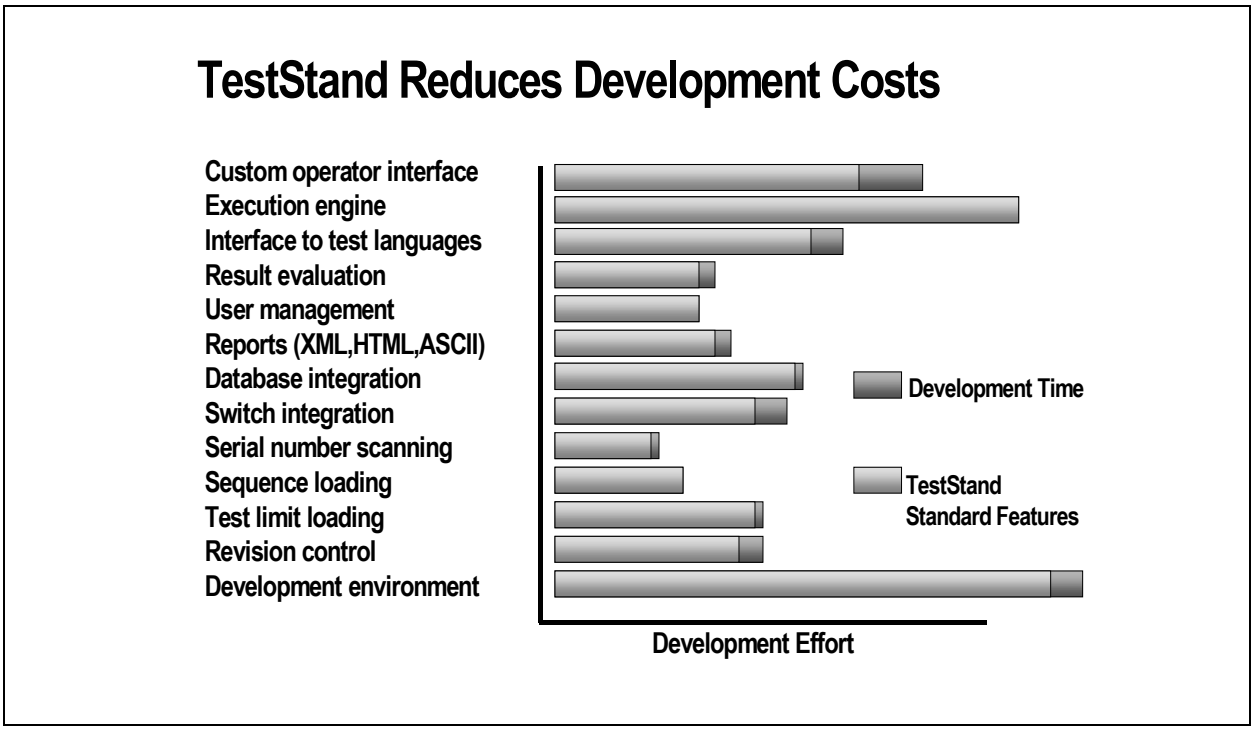
TestStand also delivers streamlined throughput by implementing intelligent decision making in the test framework that helps you ensure that Units Under Test (UUTs) only undergo tests which are appropriate. Loading test parameters dynamically at run time also streamlines throughput. This functionality enables diversity testing, where you can test a number of slightly different UUTs, such as mobile phones with different specified bandwidths, with zero downtime.

Finally, TestStand reduces overall maintenance costs, which is one of the largest hidden costs in a test system. Maintaining systems covers many areas, but one of the biggest and most avoidable issues is obsolescence. Over the past 10 years, there were at least five shifts in the use of operating systems. These shifts often cost organizations hundreds of thousands of dollars because of code rewrites, re-engineering, and lost manufacturing capability during a system upgrade. TestStand also has a wealth of training materials available to avoid unnecessary training costs.



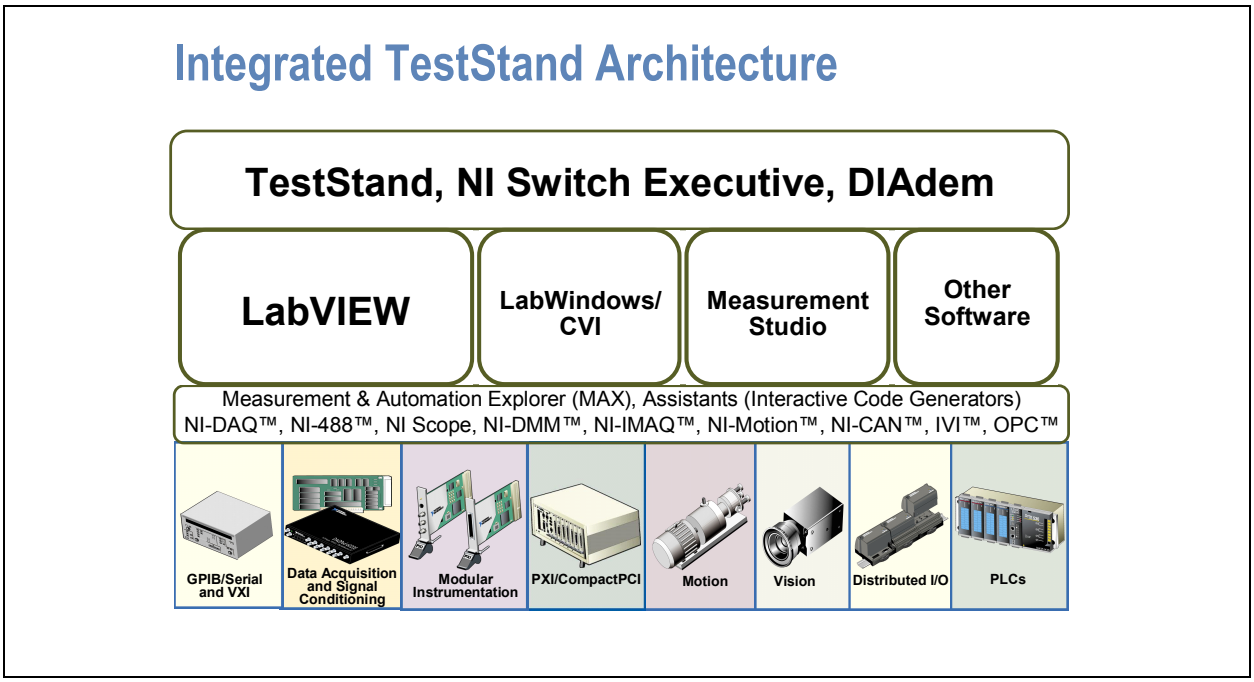
Build versus Buy Development Costs

Building custom test management software can be a daunting task. The operator interface, sequence development environment, database integration, and interface with other test development languages are just a few of the factors you must consider.



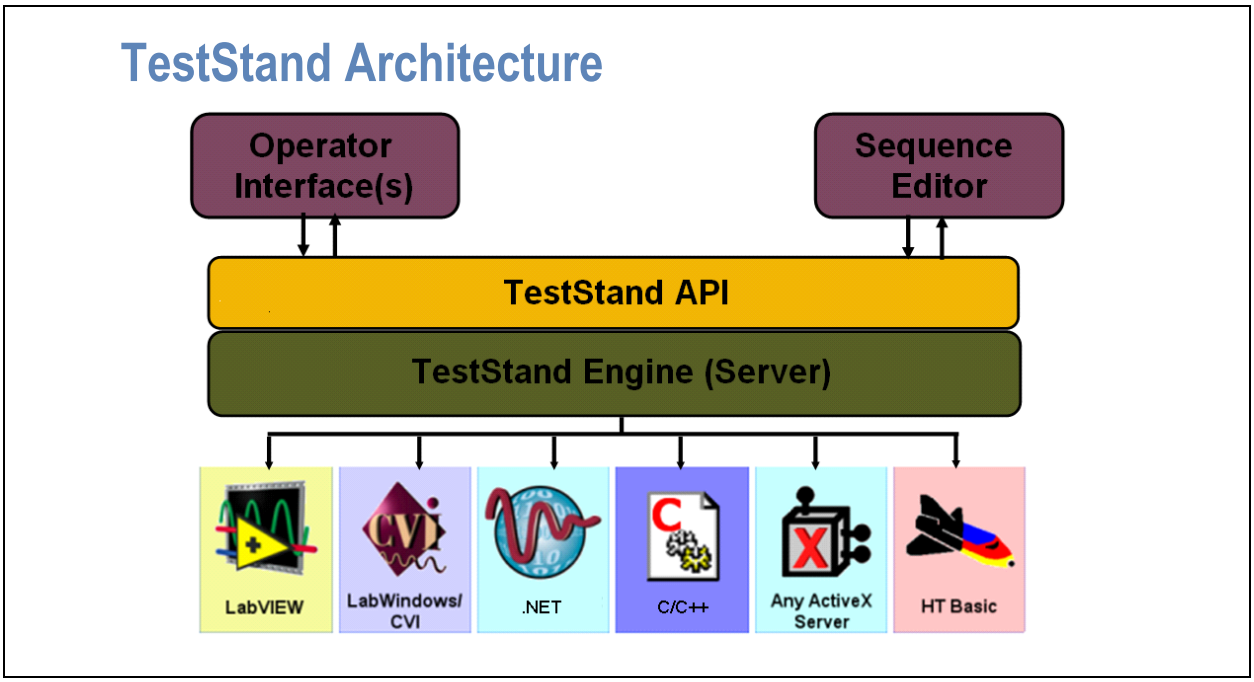
TestStand Reduces Development Costs

The features and development framework provided by TestStand fulfill many of the typical test management software specifications. Also, because each organization has different requirements for its operator interface, reporting, and so on, TestStand is specifically designed for customization and includes tools for providing the additional features you need to complete the specifications. TestStand saves you time and money by providing the core functionality you would otherwise have to design, code, maintain, and support yourself.



Integrated TestStand Architecture

Before taking a closer look at the TestStand system architecture, it is important to understand where TestStand fits into a modular test system architecture. The top layer of the National Instruments integrated test system architecture contains the software tools designed to assist you in managing large test, switching, and data analysis applications. These tools, such as TestStand, integrate directly with the other industry standard NI software tools, such as LabVIEW and LabWindows/CVI. These application development environments (ADEs) perform the actual instrument I/O and analysis of each acquired measurement through the Measurement and Control Services software, also known as the driver software. Using this framework, an organization can deliver products to market faster, achieve greater product quality, and lower development and production costs.



TestStand Architecture

The modular TestStand architecture is comprised of the following components—the TestStand Engine, Sequence Editor, Operator Interface(s), and Module Adapters.

The TestStand Engine plays a central role in the TestStand architecture. The TestStand Engine is a set of DLLs that export an extensive ActiveX Automation Application Interface (API) for creating, editing, running, and debugging sequences. The TestStand Engine handles test executive tasks such as sequencing, looping, limit checking, data allocation, and user management. The engine is optimized for speed and designed for maximum flexibility and long-term compatibility. The TestStand Sequence Editor and User Interface (UI) Controls use the TestStand API. You can access the TestStand API from any programming environment that supports access to ActiveX automation servers; thus, you can even call the TestStand API from a code module. Refer to the *TestStand Help* for more information about the TestStand API.

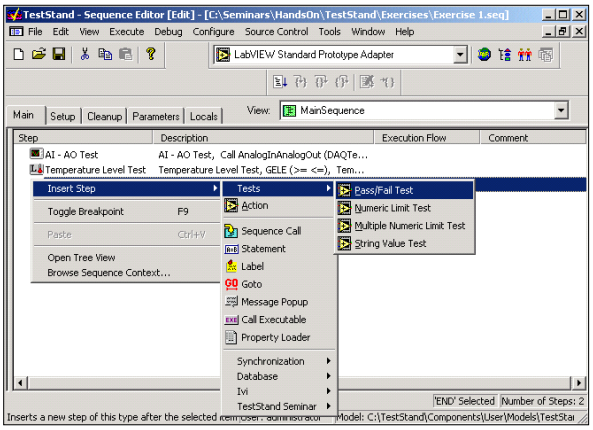
The TestStand Sequence Editor and Operator Interface(s) act as ActiveX clients and utilize the TestStand API for accomplishing such tasks as creating, editing, executing, and debugging sequences. You can call the TestStand API from any programming environment that supports access to ActiveX automation servers.

For the engine to communicate with external code modules, it must know the code module type, how to call it, and how to pass parameters to it. The TestStand module adapters act as an interface between the engine and the

external test development environments to provide this functionality. TestStand includes adapters for LabVIEW, LabWindows/CVI, C-style DLLs, ActiveX automation servers, .NET assemblies, and HTBasic.

TestStand Sequence Editor

- Create
- Edit
- Manage
- Execute
- Debug
- View reports
- Deploy
- Manage user profiles
- Customize



TestStand Sequence Editor

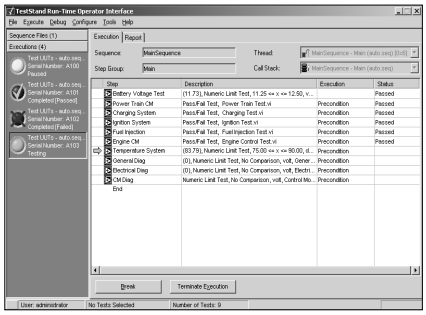
Use the TestStand Sequence Editor to create and manage test sequences and the overall test system. The TestStand Sequence Editor simplifies the creation, editing, managing, execution, and debugging of sophisticated test systems. Use the sequence editor to configure and view test reports or data stored in a database and to manage or control user access to tests. You can also use the TestStand Sequence Editor to customize many aspects of the TestStand architecture and to deploy test systems after completing development.

The TestStand Sequence Editor is an application program in which you create, modify, and debug sequences. The sequence editor gives easy access to powerful TestStand features, such as step types and process models. The sequence editor uses debugging tools that you are familiar with in application development environments (ADEs) such as LabVIEW, LabWindows/CVI, and Microsoft Visual Studio .NET. These include breakpoints, single-stepping, stepping into or over function calls, tracing, a variable display, and a Watch Expression pane.

In the TestStand Sequence Editor, you can start multiple concurrent executions, execute multiple instances of the same sequence, and execute different sequences at the same time.

TestStand Operator Interfaces

- Deployable view of test system
- Ready-to-run versions written in LabVIEW, LabWindows/CVI, Visual Basic .NET, C++, and C#
- Source code provided for customization



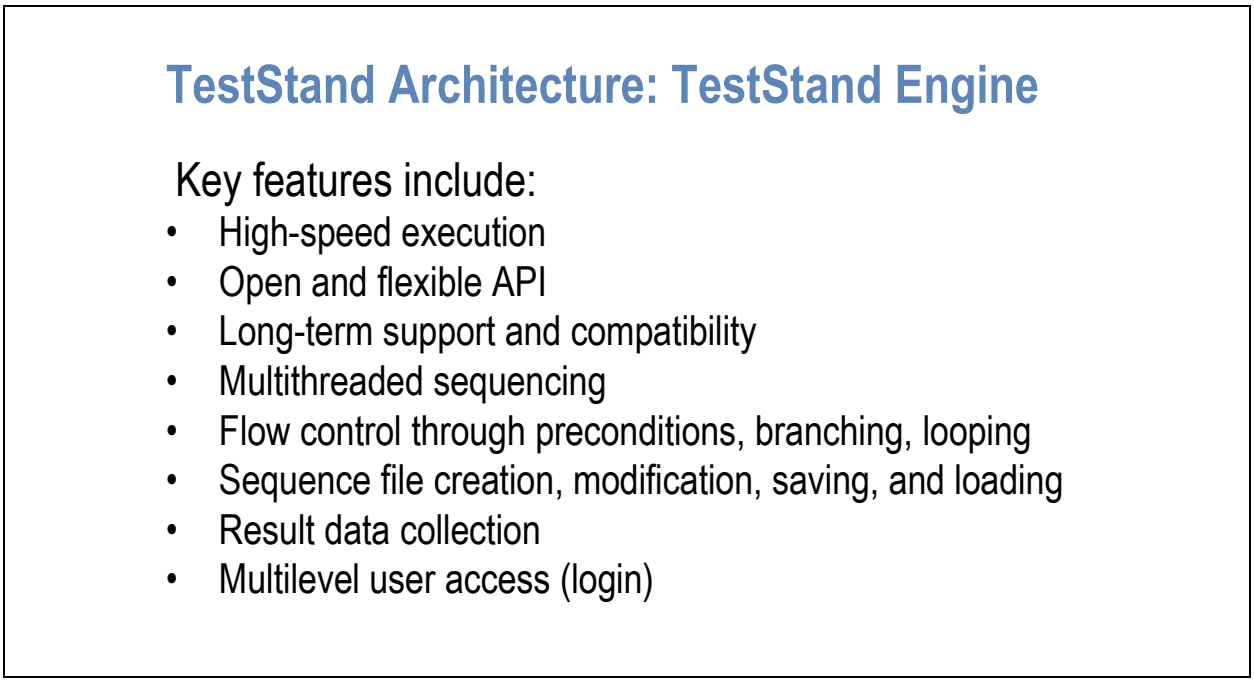
TestStand Operator Interfaces

An important aspect of any test system is how it appears to the operator in the validation lab or on the factory floor. The operator interface can vary widely among different testing needs and applications and between different types of users such as operators and technicians. Unlike the sequence editor, you cannot edit or create sequences within the operator interface. However, because the audience for the operator interface is so wide, TestStand includes operator interfaces developed in a variety of ADEs. Each operator interface is a separate application program. These interfaces—developed in LabVIEW, LabWindows/CVI, Microsoft Visual Basic .NET, C#, and C++ (MFC)—are fully customizable and are provided in both source and executable formats.

Like the TestStand Sequence Editor, the operator interfaces allow you to start multiple concurrent executions, set breakpoints, and single-step. However, the operator interfaces do not allow you to modify sequences, and they do not display sequence variables, sequence parameters, step properties, and so on. Operator interfaces are intended for use with deployed test systems and are designed to protect the integrity of your test sequences.

The TestStand architecture provides long term support of customized operator interfaces, and all the customizations. As new versions of the TestStand Engine provide increasing capability, you can plug the new versions into existing operator interfaces that you have customized.

Refer to the *TestStand II: Customization Course Manual* and the *TestStand Reference Manual* for more information about customizing operator interfaces.

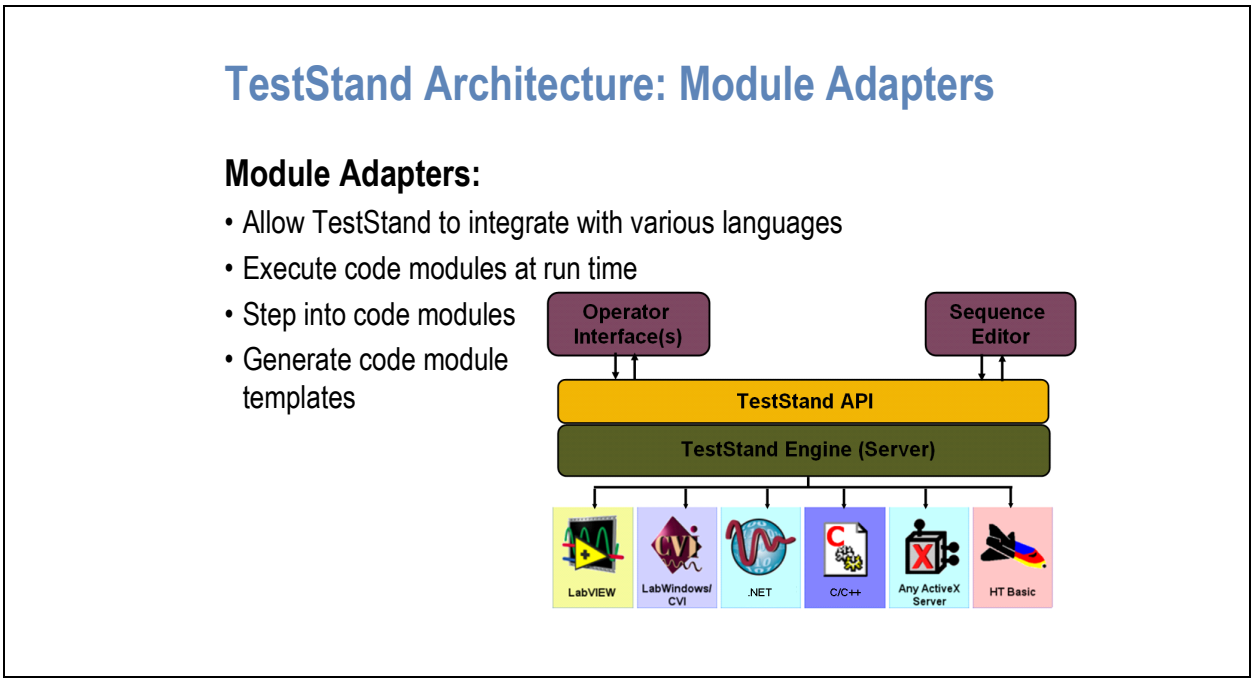


TestStand Architecture: TestStand Engine

At the core of the TestStand architecture is the TestStand Engine. The engine is implemented as a set of 32-bit Dynamic Link Libraries (DLLs) that export an ActiveX API. The TestStand Engine handles test executive tasks such as sequencing, branching, looping, limit checking, data allocation, and user management. The engine is optimized for speed and designed for maximum flexibility and long-term compatibility.

The TestStand Sequence Editor and Operator Interfaces use the TestStand API for creating, editing, executing, and debugging sequences. You can call the TestStand API from any programming environment that supports access to ActiveX automation servers. You can even call the TestStand API from code modules, including code modules you create in LabVIEW and LabWindows/CVI.

Refer to Lesson 5, *Creating Code Modules in External Environments*, of this manual for more information about ActiveX automation and the TestStand API.



TestStand Architecture: Module Adapters

Most steps in a TestStand sequence invoke code in another sequence or in a code module. When invoking code in a code module, TestStand must know the type of code module, how to call it, and how to pass parameters to it. The different types of code modules include LabVIEW VIs; C functions in source, object, or library modules created in LabWindows/CVI; C/C++ functions in DLLs; objects in .NET assemblies; objects in ActiveX automation servers; and subroutines in HTBasic. TestStand also must know the list of parameters required by the code module. TestStand uses a module adapter to obtain this knowledge. Module adapters act as an interface between TestStand and external test development environments, supplying the necessary information for communication between the two environments.

If the module adapter is specific to an ADE, the adapter knows how to 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. Module adapters allow TestStand to execute code modules, pass and receive data, generate code module templates, and even debug code written in other programming languages. TestStand module adapters allow you to integrate your existing test software and tools. Module adapters help you benefit from the fully customizable features of TestStand and the investment you have already made in your existing test software infrastructure.

Note When you create TestStand sequences, you can use different adapters within the same sequence.

Exercise 1-1: Running a Sequence File

Objective: To run test sequences from the sequence editor and view reports that TestStand generates.

Estimated Time: 10 minutes

Refer to page 1-18 for instructions for this exercise.

Exercises 1-2A: Running a Sequence File from the LabVIEW Operator Interface

Exercise 1-2B: Running a Sequence File from the LabWindows/CVI Operator Interface

Objective: To run a test sequence file from the LabVIEW or LabWindows/CVI Operator Interface and view reports that TestStand generates.

Estimated Time: 10 minutes

Note When an exercise number includes A or B, select the exercise that uses your preferred development environment.

Refer to page 1-22 for instructions for Exercise 1-2A.

Refer to page 1-26 for instructions for Exercise 1-2B.

Lesson 1 Summary: Introduction to TestStand

- **Test management software provides an open, flexible framework for managing common test system tasks**
- **TestStand is an off-the-shelf, ready-to-run test management environment that is fully customizable**
- **TestStand has four main components:**
  - Sequence Editor
  - Operator Interfaces
  - TestStand Engine
  - Module Adapters

Summary

A test management environment performs tasks in the testing environment that are repetitive for different products tested. TestStand allows you to focus on creating the code modules and prevents redundant programming.

TestStand is ready-to-run test management software that is fully customizable. This means you can modify TestStand to fit your needs.

The TestStand architecture includes the following four main components:

- **Sequence Editor**—Used for developing, debugging, and executing sequences.
- **Operator Interface(s)**—Used for executing and debugging sequences on the production floor.
- **TestStand Engine**—An ActiveX server that performs the test executive operations.
- **Module Adapters**—Enables TestStand to interface with external development environments.

Lesson 1 Summary (Continued)

Sequence Editor and Operator Interface Comparison

	Sequence Editor	Operator Interface
Modify/create sequences	Yes	No
Debug sequences (trace, breakpoints, and single-stepping)	Yes	Yes
Customizable	No	Yes
Multiple concurrent executions	Yes	Yes
View/modify properties	Yes	No

Summary (Continued)

Similarities and Differences Between the Sequence Editor and Operator Interface

Both the operator interface and the sequence editor are separate applications. These applications are both ActiveX clients to the TestStand Engine but contain similarities and differences that make them useful for different purposes.

Some of the similarities between the two applications are that both allow the user to complete the following tasks:

- Run multiple, concurrent executions.
- Debug sequences using debugging tools, such as single-stepping, setting breakpoints, enabling tracing, and others.

The two main differences between the two applications include:

- You cannot customize the appearance of the sequence editor. However, all of the source code for the operator interfaces is provided, which allows you to customize any component of the operator interfaces.
- You cannot modify or create sequences from your operator interface. All development of sequences, including displaying variables, inserting steps, viewing and modifying properties, and so on, must be performed with the sequence editor.

Although you can use the TestStand Sequence Editor at a production station, the TestStand Operator Interfaces are simpler and fully customizable, thus making them more suitable applications for deployment purposes.

Exercise 1-1 Running a Sequence File

Objective: To run test sequences from the sequence editor and view the report TestStand generates.

The sequence editor is the TestStand development environment you use to create, edit, and execute sequences. It is important to be familiar with this environment because you use it to develop all sequence files.

1. Select **Start»Programs»National Instruments»TestStand»Sequence Editor**, to open the sequence editor,
2. After the sequence editor loads, the Login dialog box opens, as shown in Figure 1-1. Select the default, `administrator`, from the **User Name** control and leave the **Password** text box empty.
3. Click **OK** in the Login dialog box.

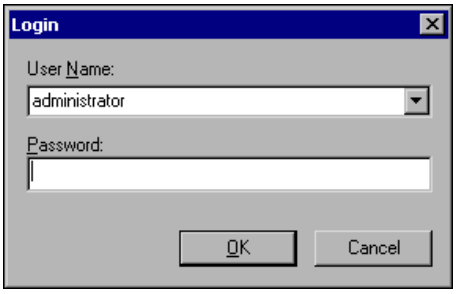


Figure 1-1. Login Dialog Box

4. Select **File»Open** and navigate to the `C:\Exercises\TestStand I` directory.
5. Double-click the `Ex 1-1 Computer Test.seq` file to open the sequence file. Figure 1-2 shows the TestStand window that opens.

Note All *TestStand I: Introduction* course exercises are located in the `C:\Exercises\TestStand I` directory.

Additional Information

The `Ex 1-1 Computer Test.seq` sequence simulates testing parts of a computer motherboard such as the RAM, ROM, and Keyboard tests. This sequence has tests written in LabVIEW; however, you could write the tests in LabWindows/CVI or a variety of other programming environments compiled as DLLs, EXEs, or ActiveX servers.

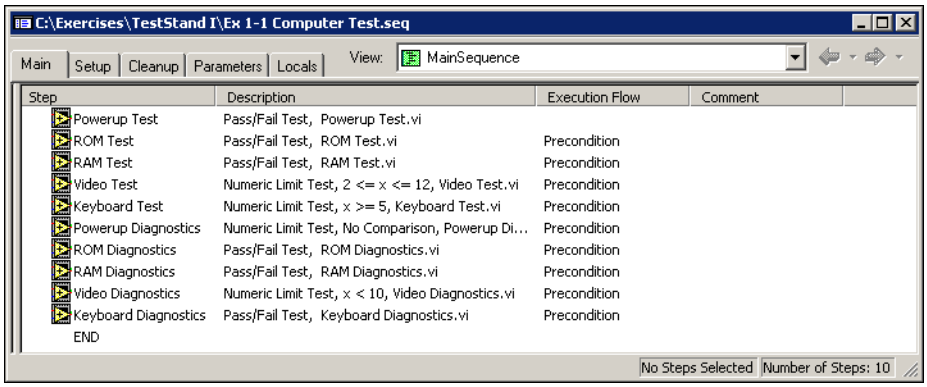


Figure 1-2. Computer Test Sequence File

6. Select **Execute»Test UUTs** or click **Run**, at left of the tool bar, to run the sequence.

   Additional Information

   Use Execution entry points, available from the Execute menu, to run the sequence in predefined ways. By default, TestStand has two Execution entry points, Test UUTs and Single Pass. Refer to Lesson 2, *TestStand Environment*, for more information about Execution entry points and how to create them.

   The Test UUTs Execution entry point runs the sequence in a continuous loop. Select **Execute»Test UUTs** to run most test systems in TestStand.

   The Single Pass Execution entry point executes the sequence once and generates a report. Select **Execute»Single Pass** to debug tests and determine that a sequence execution proceeds as you intended. Refer to Lesson 2, *TestStand Environment*, for more information about debugging tools.

   You can also select **Execute»Run MainSequence** to execute the sequence without using the process model. This mode can only test how the `MainSequence` proceeds, because no other parts of the process model execute in this type of execution. For now, the only noticeable difference is that no report generates, as compared to the Test UUTs or Single Pass entry points. Refer to Lesson 2, *TestStand Environment*, for more information about the process models used by the Test UUTs and Single Pass entry points.

7. A UUT Information dialog box launches prior to executing the sequence, as shown in Figure 1-3. Enter any number for the serial number and click **OK** to continue.

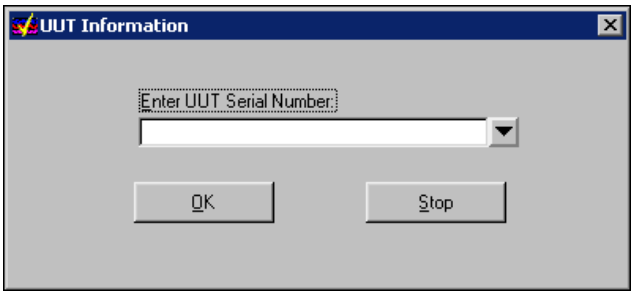


Figure 1-3. UUT Information Dialog Box

8. The Test Simulator dialog box shown in Figure 1-4 launches. Use this dialog box to enable the tests you want to fail during testing. To enable a test to fail, click the box next to that test. An **X** appears in the box that you click. Select the tests you want to fail, then click **OK** to execute the sequence.

Note Enabling the CPU test in the Test Simulator dialog box does not cause a failure. The CPU Test step is not yet included in this sequence. You will add this step to the sequence later in the course.

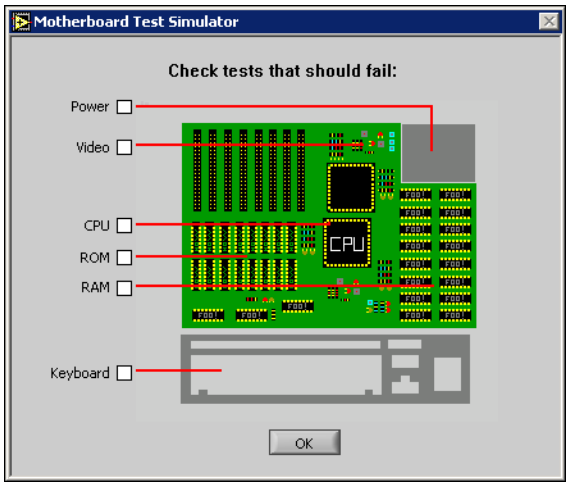
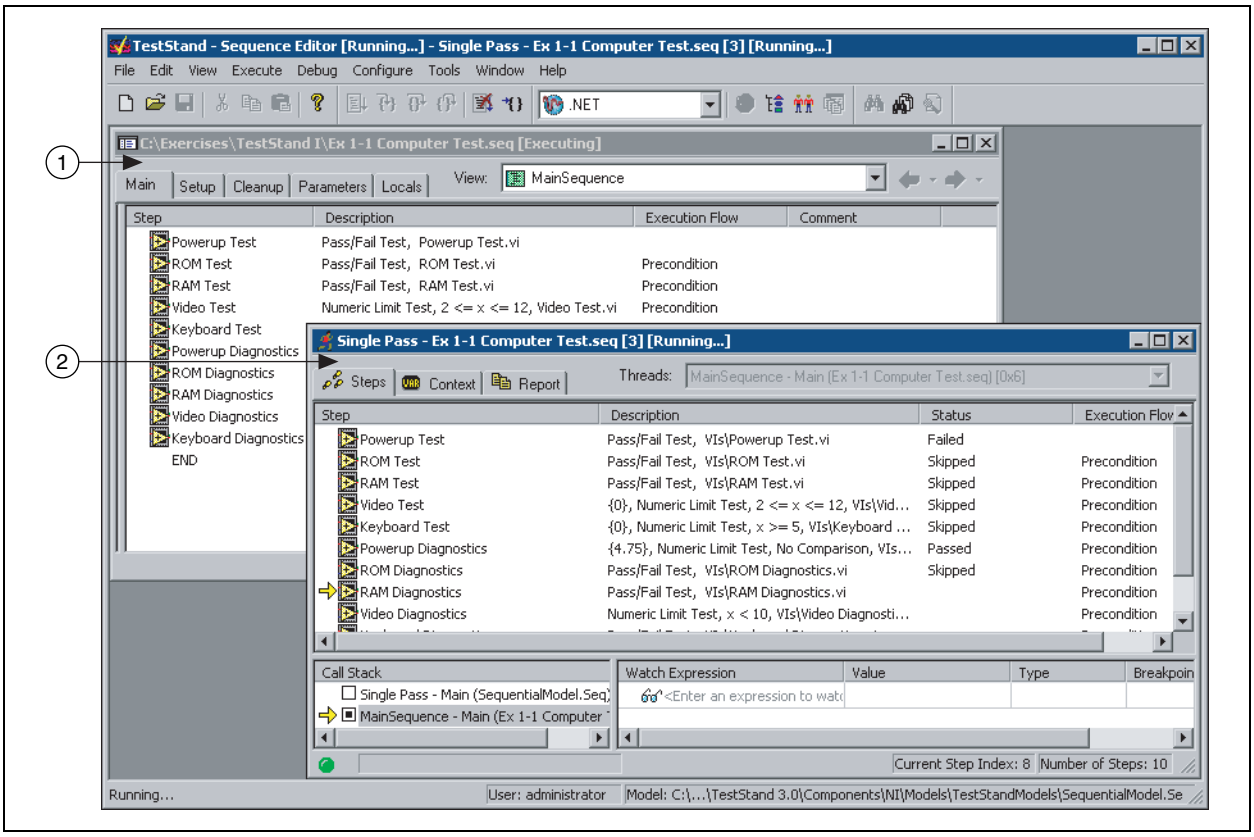


Figure 1-4. Test Simulator Dialog Box

9. After executing the sequence, a UUT Result dialog box opens, indicating the status of the sequence. Click **OK** to close the dialog box.
10. The UUT Information dialog box opens again. To stop execution and generate a test report, click **Stop**.

    Notice that another window launches in the sequence editor when the sequence begins execution. This is the Execution window. The window that initially appeared was the Sequence File window. Figure 1-5 shows both windows.



1 Sequence File Window	2 Execution Window

Figure 1-5. TestStand Windows

11. After the execution completes, the Execution window displays the generated test report. Scroll down the test report and verify which test failed.

12. Select **Execute»Test UUTs** again and run through several UUTs, setting different tests to fail each time to become familiar with the execution of this sequence.

    Additional Information

    This sequence uses preconditions to determine when tests execute. For example, if you set the Powerup test to fail, no other tests execute. Diagnostic steps only execute if their corresponding test fails. This is a common scenario in test systems. If a test fails, you might want to run a diagnostic test to determine which component failed, but you do not need to run a diagnostic test if the original test passed. Dependencies that determine which steps execute are called preconditions.

13. Select **File»Close** twice to close the Execution and Sequence File windows.

14. Select **File»Exit** to close the sequence editor.

End of Exercise 1-1

Exercise 1-2A Running a Sequence File from the LabVIEW Operator Interface

Objective: To run a test sequence file from the operator interface and view reports that TestStand generates.

In addition to the sequence editor application to execute a test in the previous exercise, you can also use the operator interface to execute tests. The operator interface in this exercise is built with LabVIEW and is the interface the operator uses on the production floor.

The operator interface is similar to the sequence editor because it has most of the same functionality. The operator interface and the sequence editor have the following differences:

- The sequence editor can modify or create sequence files. The operator interface cannot modify or create sequence files.
- You cannot modify the sequence editor because the source code is not provided. Because the source code for the operator interface installs with TestStand, you can modify the code and create a customized operator interface for every user.

Note The source code for the default operator interfaces is located in the `<TestStand>\OperatorInterfaces` directory.

This exercise explores the default LabVIEW Operator Interface so you can become familiar with some of the differences and similarities between it and the sequence editor.

1. Select **Start»Programs»National Instruments»TestStand»Operator Interfaces»LabVIEW** to open the operator interface.

   Additional Information
   The operator interface is an application for opening and running test sequences. The sample operator interfaces included with TestStand are intended to be a starting point for the application that you install on the production floor. The operator interfaces allow the execution and debugging of multiple sequences.

2. After the LabVIEW Operator Interface loads, the Login dialog box displays. Select the default, `administrator`, from the **User Name** control and leave the **Password** text box empty.
3. Click **OK** in the Login dialog box.
4. The Sequence File window opens in the operator interface. Select **File» Open Sequence File** and open the `Ex 1-1 Computer Test.seq` file located in the `C:\Exercises\TestStand I` directory.

This is the same sequence that you ran in Exercise 1-1. Figure 1-6 shows the sequence file displayed in the operator interface.

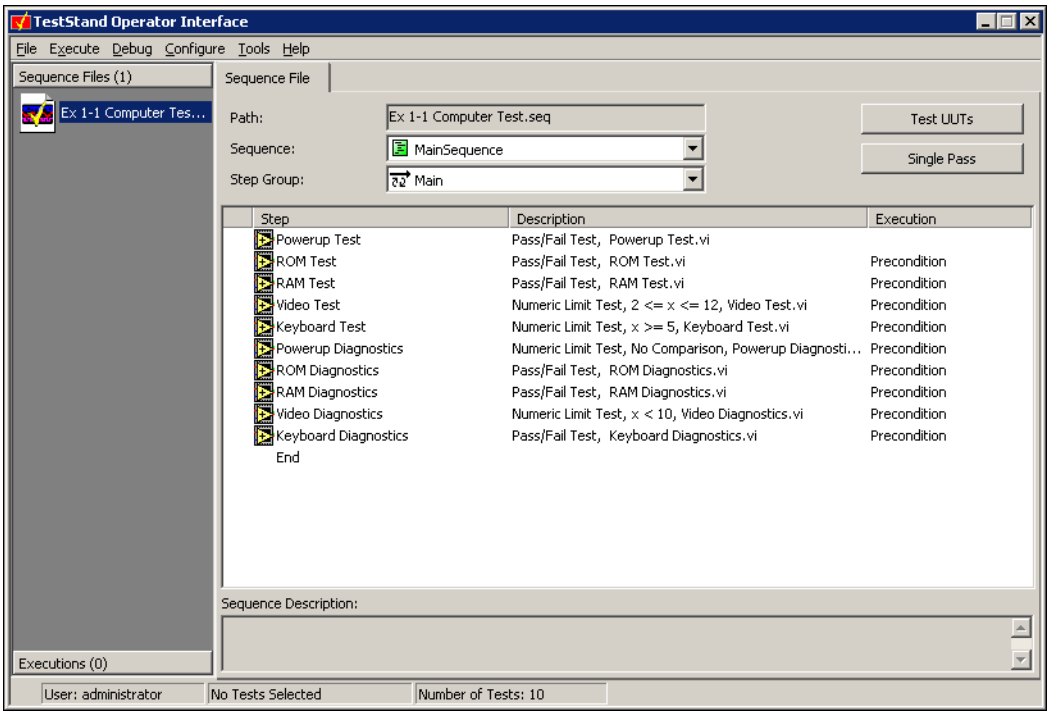


Figure 1-6. LabVIEW Operator Interface

5. Click **Test UUTs** to execute the sequence. The Execution window opens in the operator interface and the UUT Information dialog box launches.
6. Enter any value for the serial number in the UUT Information dialog box and click **OK** to start the sequence.
7. Select different tests to fail in the Test Simulator dialog box and click **OK** to run the sequence.

   Additional Information

   During execution, the Execution window displays the status of each step and allows you to control the execution using tools such as breakpoints and stepping mechanisms. Refer to Lesson 2, *TestStand Environment*, for more information about breakpoints and stepping tools.
8. Click **Stop** in the UUT Information dialog box to stop the execution. The report displays in the Execution window.

   Figure 1-7 shows the report displayed in HTML, the default display format for the LabVIEW Operator Interface.

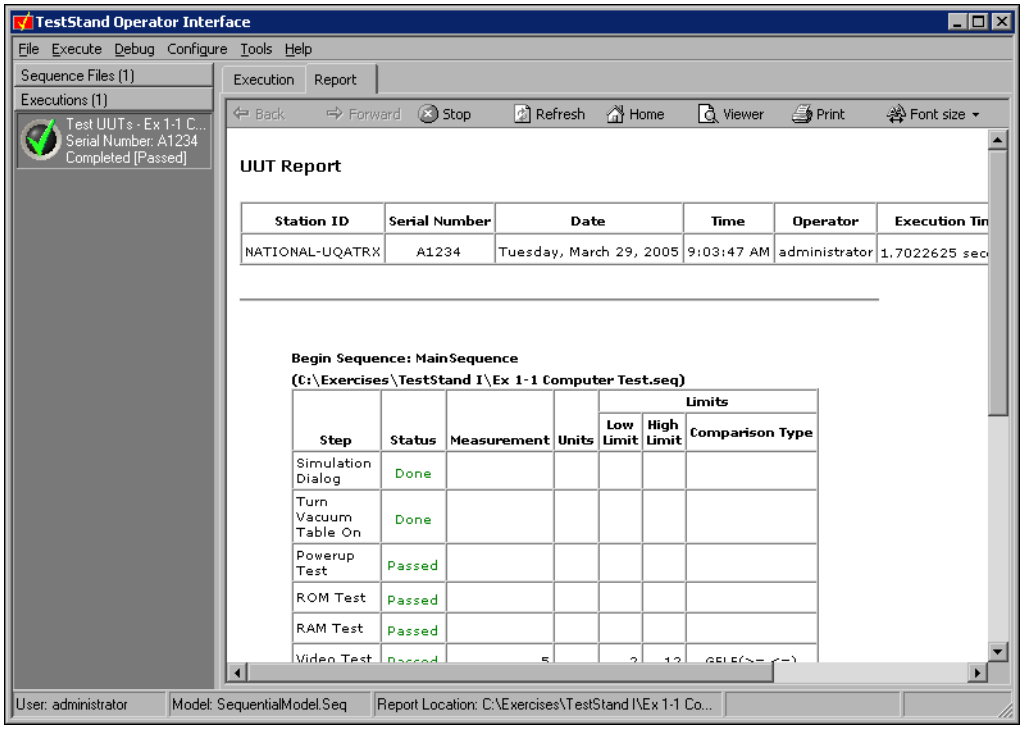


Figure 1-7. LabVIEW Operator Interface Execution View

Additional Information

The Sequence File and Execution windows **are separate views of the same program in the operator interface**. The Sequence File window displays all information related to the sequence file(s) currently loaded into memory including the sequence filename, the current logged in user, and the steps in the sequence. The Execution window displays all the information related to the execution of a sequence file including the status of each step in the sequence file, which sequence is running, and the sequence report contents. Click the name of either view in the left pane of the operator interface to bring it to the front and display its contents.

Note TestStand can run multiple sequence executions in parallel. Switching back to the Sequence File window and clicking the **Test UUTs** button again causes the sequence to begin a second execution in a new Execution window. All loaded sequence files and executions display in the left pane of the operator interface. To change the view to a different sequence or execution, click the view in the left pane, then click the file or execution you want to display. You can also load and execute a separate sequence file in parallel instead of executing the same sequence file. Refer to Lesson 3, *Creating Sequences*, for an example of executing sequences in parallel.

9. Click the **Report** tab in the Execution window to display the report.
10. Select **File»Close All Sequence Files and Executions**.
11. Select **File»Exit** to close the operator interface.

End of Exercise 1-2A

Exercise 1-2B Running a Sequence File from the LabWindows/CVI Operator Interface

Objective: To run a test sequence file from the operator interface and view reports that TestStand generates.

In addition to the sequence editor application you used to execute a test in the previous exercise, you can also use the operator interface to execute tests. The operator interface in this exercise was built with LabWindows/CVI and is the interface the operator uses on the production floor.

The operator interface is similar to the sequence editor because it has most of the same functionality. The operator interface and the sequence editor have the following differences:

- The sequence editor can modify or create sequence files. The operator interface cannot modify or create sequence files.
- You cannot modify the sequence editor because the source code is not provided. Because the source code for the operator interface installs with TestStand, you can modify the code and create a customized operator interface for every user.

Note The source code for the default operator interfaces is located in the `<TestStand>\OperatorInterfaces` directory.

This lesson explores the default LabWindows/CVI Operator Interface so you can become familiar with some of the differences and similarities between it and the sequence editor.

1. Select **Start»Programs»National Instruments»TestStand»Operator Interfaces»LabWindows-CVI** to open the operator interface.

   Additional Information
   The operator interface is an application for opening and running test sequences. The sample operator interfaces included with TestStand are intended to be a starting point for the application that you install on the production floor. The operator interfaces allow the execution and debugging of multiple sequences.

2. After the LabWindows/CVI operator interface loads, the Login dialog box displays. Select the default, `administrator`, from the **User Name** control and leave the **Password** text box empty.
3. Click **OK** in the Login dialog box.
4. The Sequence File window opens in the operator interface. Select **File»Open Sequence File** and open the `Ex 1-1 Computer Test.seq` file located in the `C:\Exercises\TestStand I` directory.

This is the same sequence that you ran in Exercise 1-1. Figure 1-8 shows the sequence file displayed in the operator interface.

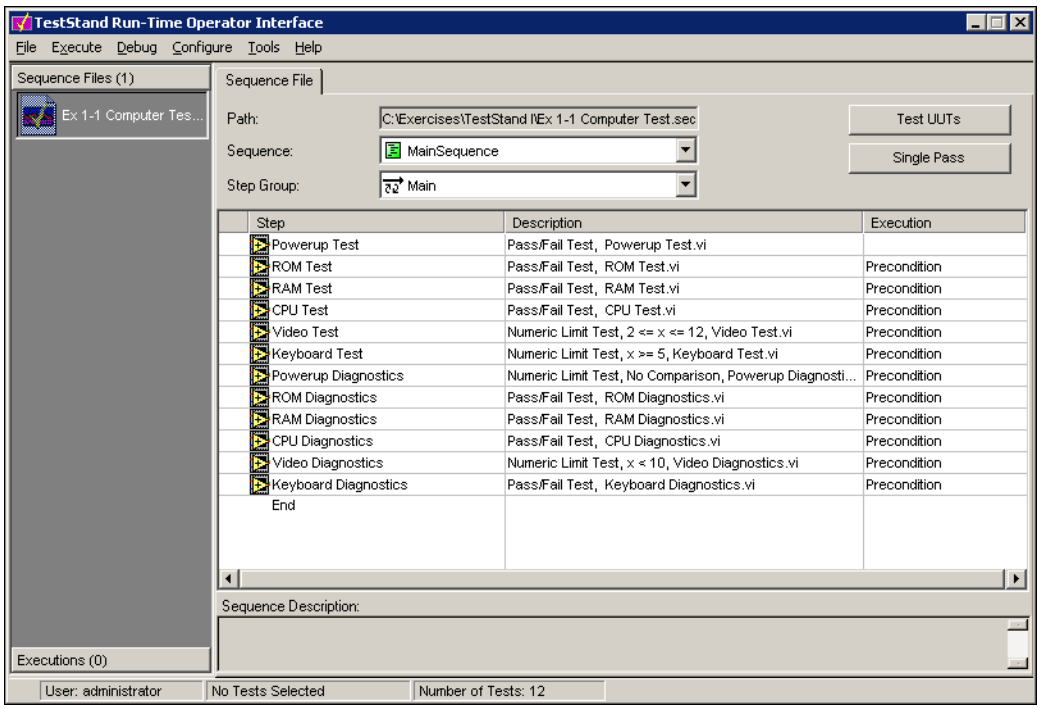


Figure 1-8. LabWindows/CVI Operator Interface

5. Click **Test UUTs** to execute the sequence. The Execution window opens in the operator interface and the UUT Information dialog box launches.
6. Enter any value for the serial number in the UUT Information dialog box and click **OK** to start the sequence.
7. Select different tests to fail in the Test Simulator dialog box and click **OK** to run the sequence.

   Additional Information

   During execution, the Execution window displays the status of each step and allows you to control the execution using tools such as breakpoints and stepping mechanisms. Refer to Lesson 2, *TestStand Environment*, for more information about breakpoint and stepping tools.
8. Click **Stop** in the UUT Information dialog box to stop the execution. The report displays in the Execution window.

   Figure 1-9 shows the report displayed in HTML, the default display format for the LabWindows/CVI Operator Interface.

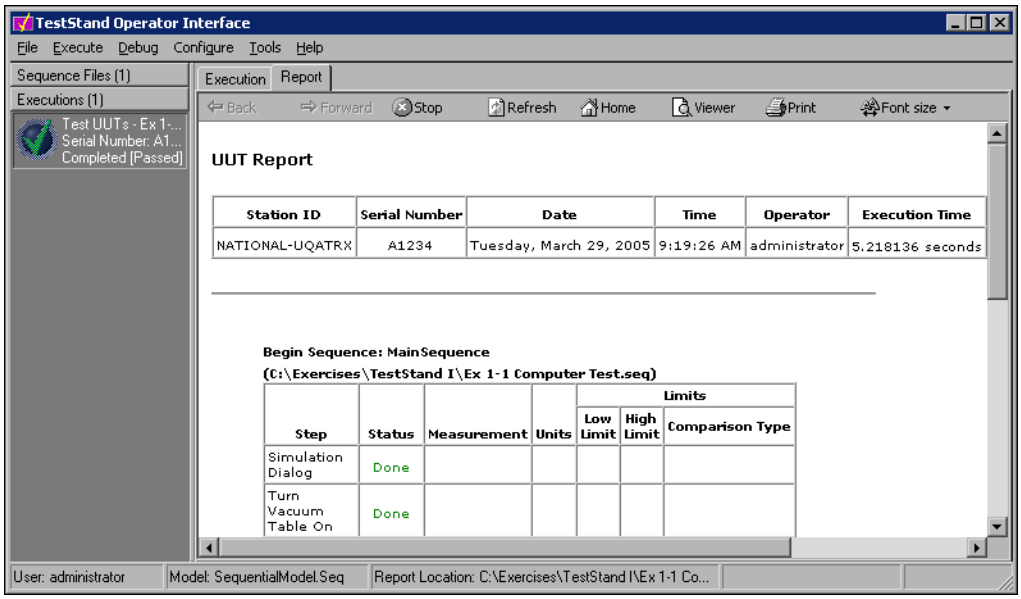


Figure 1-9. LabWindows/CVI Operator Interface Execution View

Additional Information

The Sequence File and Execution windows are separate views of the same program in the operator interface. The Sequence File window displays all information related to the sequence file(s) currently loaded into memory including the sequence filename, the current logged in user, and the steps in the sequence. The Execution window displays all the information related to the execution of a sequence file including the status of each step in the sequence file, which sequence is running, and the sequence report contents. Click the name of either view in the left pane of the operator interface to bring it to the front and display its contents.

Note TestStand can run multiple sequence executions in parallel. Switching back to the Sequence File window and clicking the **Test UUTs** button again causes the sequence to begin a second execution in a new Execution window. All loaded sequence files and executions display in the left pane of the operator interface. To change the view to a different sequence or execution, click the view in the left pane, then click the file or execution you want to display. You can also load and execute a separate sequence file in parallel instead of executing the same sequence file. Refer to Lesson 3, *Creating Sequences*, for an example of executing sequences in parallel.

9. Click the **Report** tab in the Execution window display the report.
10. Select **File»Close All Sequence Files and Executions**.
11. Select **File»Exit** to close the operator interface.

End of Exercise 1-2B

Self Review

1. In general, explain the role of a test executive.

2. Explain why TestStand is considered two products in one.

3. List three ways you can use TestStand to lower your cost of testing.

4. List the four main components of TestStand, and give a brief explanation of each.

5. Which of the four components you listed in question 4 are ActiveX servers and which are ActiveX clients?

6. What are the differences between the sequence editor and the operator interface?

7. Explain what the TestStand API is.

8. Draw a diagram that demonstrates the architecture of TestStand.

Notes

2

TestStand Environment

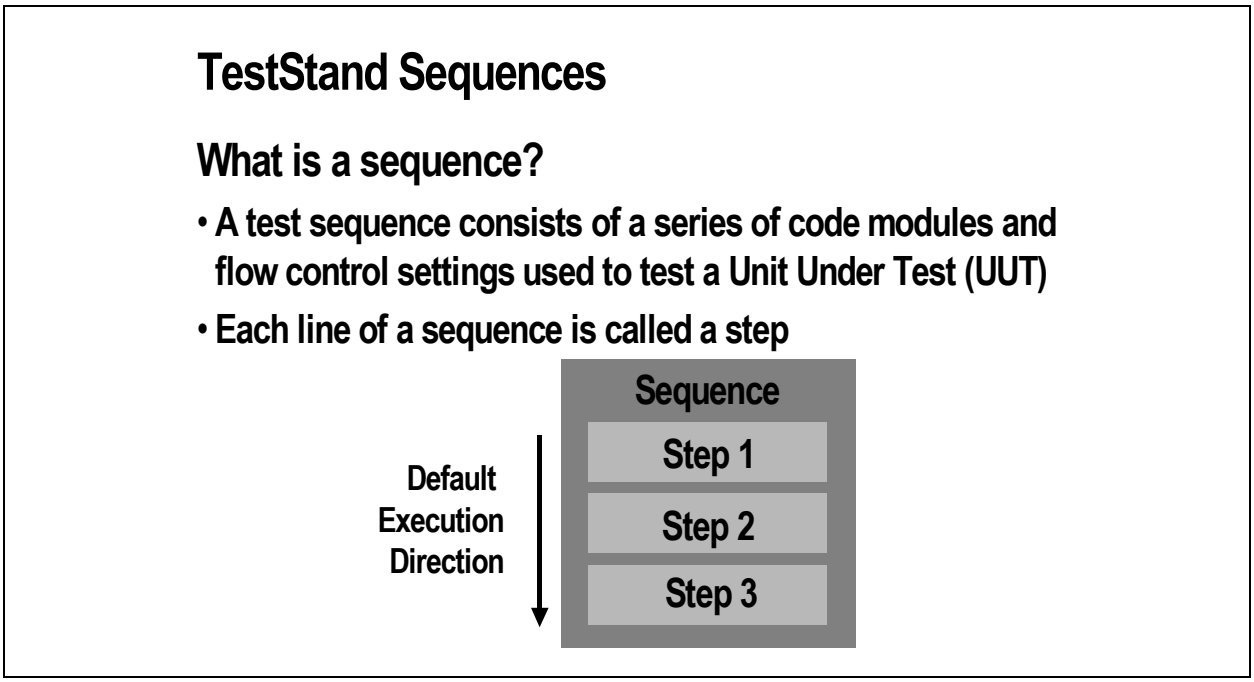
Lesson 2: TestStand Environment

In this lesson, you will learn about:

- TestStand sequences, step groups, and sequence files
- Debugging features in TestStand and how to use them
- Process model and the default TestStand process models
- Features available in the sequence editor for managing test development

Introduction

This lesson discusses TestStand sequence files, sequences, and their step groups. It then introduces the debugging tools available in TestStand and explains how to use them in the sequence editor. The lesson defines the concept of a process model and explains how TestStand can use the different default process models to execute tests. Test file management tools are also discussed.

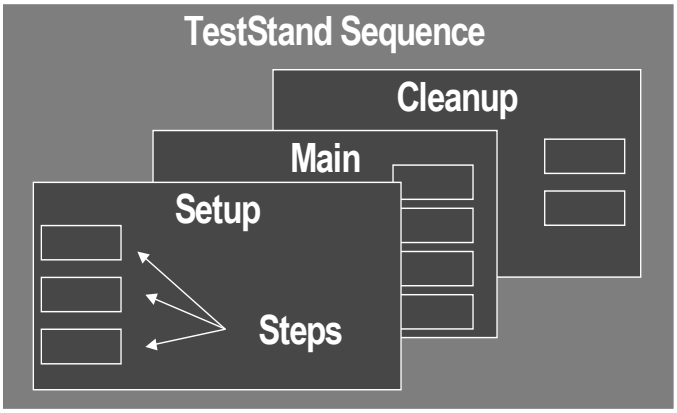


TestStand Sequences

A sequence is a series of TestStand steps and the control logic that determines the order in which those steps execute. Each step performs a particular action that contributes to the overall test of the UUT. For example, a step might execute a LabVIEW VI that performs some type of test. By default, TestStand begins executing sequences with the top most step and executes the steps from top to bottom. However, you can control the sequence execution flow. Refer to Lesson 3, *Creating Sequences*, for more information about controlling sequence execution flow.

TestStand Sequences: Step Groups

Every TestStand sequence has three step groups.



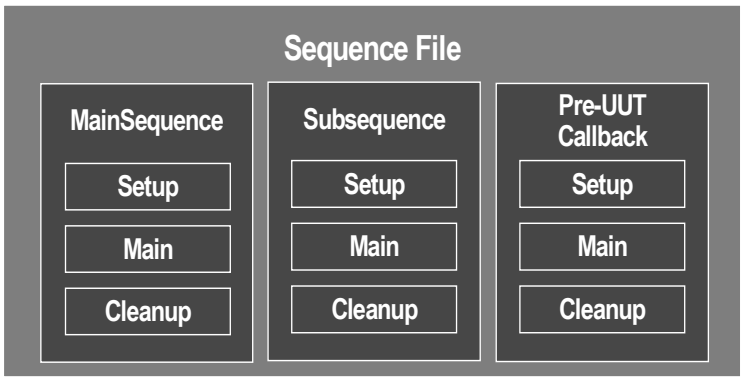
Step Groups

Every TestStand sequence contains the following three step groups: Setup, Main, and Cleanup. The step groups always execute in the same order: Setup, Main, then Cleanup. The step groups organize a series of steps in a logical and sequential manner. For example, the Setup step group might contain initialization or power-up routines, the Main step group might contain the actual test routines, and the Cleanup step group might contain steps to close files or power-down instruments.

In addition to organizing steps, step groups serve a functional role. By default, the steps in the Cleanup step group execute when the sequence terminates. Termination might result from run-time error, operator input, or programmatic control using the TestStand API methods. The steps of the Setup and Cleanup step groups execute by default when steps of the Main step group are executed interactively.

TestStand Sequence Files

Sequence files include a Main sequence. They also can include user-defined sequences and predefined sequences.



TestStand Sequence Files

TestStand sequence files are comprised of multiple sequences. The sequences might be multiple user-defined sequences and/or any callback sequences. For now, think of callbacks as predefined sequences.

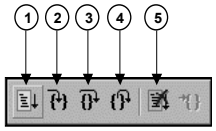
A sequence file in TestStand is an application file that contains all the information necessary for TestStand to load and execute a particular test application.

Note TestStand sequences typically contain a sequence named `MainSequence`, which is different from the Main step group that every sequence contains.

Debugging Sequences

Types of debugging tools:

- **Execution tracing**
- **Breakpoint setting**
- **Watch Expression pane**
- **Single stepping tools**



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

Debugging Tools

TestStand includes a variety of debugging tools to help you verify the execution of TestStand applications.

The execution tracing option allows you to view the execution flow for the sequence in the Execution window. If tracing is enabled, the sequence editor displays the progress of an execution by placing a yellow pointer icon to the left of the icon for the currently executing step. You can adjust the speed of tracing by selecting **Configure»Station Options** and adjusting the speed slider control in the **Execution** tab.

The sequence editor and operator interfaces allow you to set breakpoints, step into, step over, or step out of sequences, and set the next step to execute. The sequence editor also contains a Watch Expression pane that displays the values of currently active variables and expressions. These debugging tools allow you to control the sequence execution manually and identify potential areas that require further modification. Refer to Exercises 2-1 and 2-2 for more information about debugging tools.

Exercise 2-1: Understanding the Sequence File

Objective: To understand subsequences, setup and cleanup routines, and execution tracing.

Estimated Time: 15 minutes

Refer to page 2-20 for instructions for this exercise.

Exercise 2-2: Running a Sequence File with Breakpoints and Single Stepping

Objective: To become familiar with using breakpoints and debugging tools in the sequence editor.

Estimated Time: 5 minutes

Refer to page 2-25 for instructions for this exercise.

What is a Process Model?

A process model has the following characteristics:

- Specifies the high-level flow of execution in a test system
- Is associated with a client sequence file

What is a Process Model?

A process model is a list of steps that specify the high-level flow of execution in a test system, including actions that take place before and after the sequence executes, such as serial number inquiry and report generation. In addition, the process model controls the flow of successive UUT testing. The process model is a sequence file that contains the steps and sequences that the TestStand Engine uses to determine its flow of execution. The sequences contained in the process model are marked as Configuration entry points, Execution entry points, normal sequences, or Model callbacks.

By default, when you execute a test sequence using Single Pass or Test UUTs, the test sequence file becomes the client sequence file of the process model. The process model then performs a set of operations before and after executing the test sequence. In other words, when you execute the test sequence using a Process Model entry point, the test sequence does not run directly. Instead, the test sequence executes as the client sequence file of the process model.

A process model also defines a set of Execution entry points. Each entry point is a sequence in the process model file. Multiple entry points in a process model give you different options for invoking test sequences.

For example, the default TestStand process model, the Sequential model, provides two Execution entry points for invoking test sequences: Test UUTs and Single Pass. The Test UUTs entry point initiates a loop that repeatedly identifies and tests UUTs. The Single Pass entry point tests a single UUT without identifying it. Both entry points allow you to log results and produce reports.

What is a Process Model? (Continued)

- **Contains common operations of the test system**
  - Serial number entry
  - Generating reports
  - Logging to databases
  - Displaying Pass/Fail banners
  - Successive UUT Testing
- **Contains configurations of these operations**
  - Report options
  - Database options
  - Model options

What is a Process Model? (Continued)

Testing a UUT requires more than just executing a set of tests. The test management system must perform a series of operations before and after the test sequence executes in order to handle common test system tasks such as identifying the UUT, notifying the operator of the pass/fail status, generating a test report, and logging results. These operations define the testing process. The set of such operations and their flow of execution is called a process model.

The process model contains a chain of steps and sequences that perform a series of operations before and after testing the UUT. For example, the PreUUT Callback is part of the process model and is responsible for gathering the serial numbers of the UUTs. As another example, the generation of the test report is handled by the TestReport Callback, which is also located inside the process model. The process model performs both of these operations outside the client sequence file.

In addition to these operations, the process model also contains special sequences called Configuration entry points that configure certain features of the process model.

TestStand Process Models

TestStand includes the following process models:

- **`SequentialModel.seq`** (Default Process Model)—Multi-purpose and ideal for most testing applications.
- **`BatchModel.seq`**—Used to control a set of test sockets that test multiple UUTs as a group or "batch."
- **`ParallelModel.seq`**—Used for handling multiple, independent test sockets.

Note All process model sequence files are located in the `<TestStand>\Components\NI\Models\TestStandModels` directory.

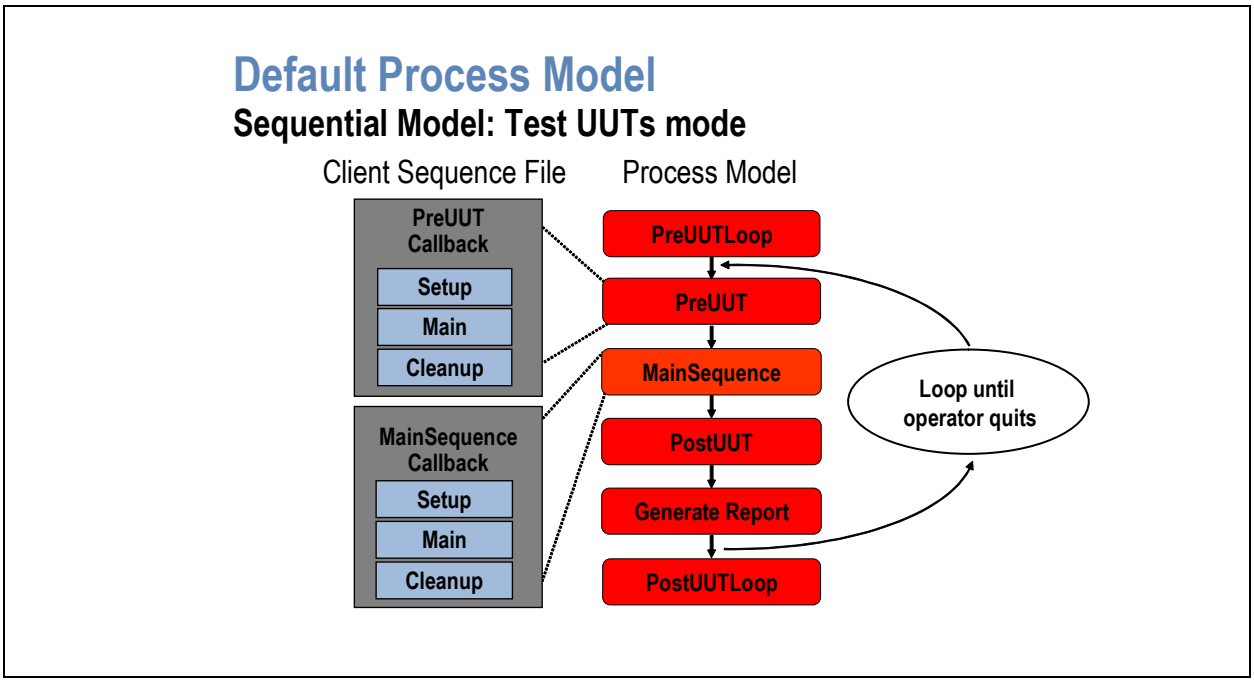
TestStand Process Models

TestStand provides a mechanism for defining a process model in the form of a sequence file. You can edit a process model just as you edit other sequence files. TestStand ships with three fully functional process models: the Sequential model, the Batch model, and the Parallel model. You can use the Sequential model to run a test sequence on one UUT at a time. The Parallel and Batch models allow you to run the same test sequence on multiple UUTs at the same time.

Use the Batch process model to control a set of test sockets that test multiple UUTs as a group. For example, you might have a set of circuit boards attached to a common carrier. The Batch process model ensures that you start and finish testing all the circuit boards in unison. The Batch process model also provides batch synchronization features you can use to configure a step or group of steps to execute in a particular manner with respect to the batch as a whole. For example, you can configure a step to run once per batch instead of once for every UUT. You also could configure the step to execute on only one UUT at a given time instead of executing in parallel. The Batch process model also can generate batch reports that summarize the result for all UUTs tested in a batch.

Use the Parallel process model to handle multiple, independent test sockets. The Parallel process model allows you to start and stop testing on any socket at any time while other sockets continue. For example, you might have five test sockets for testing radios. The Parallel process model allows you to load a new radio into an open test socket while the other sockets are busy testing other radios.

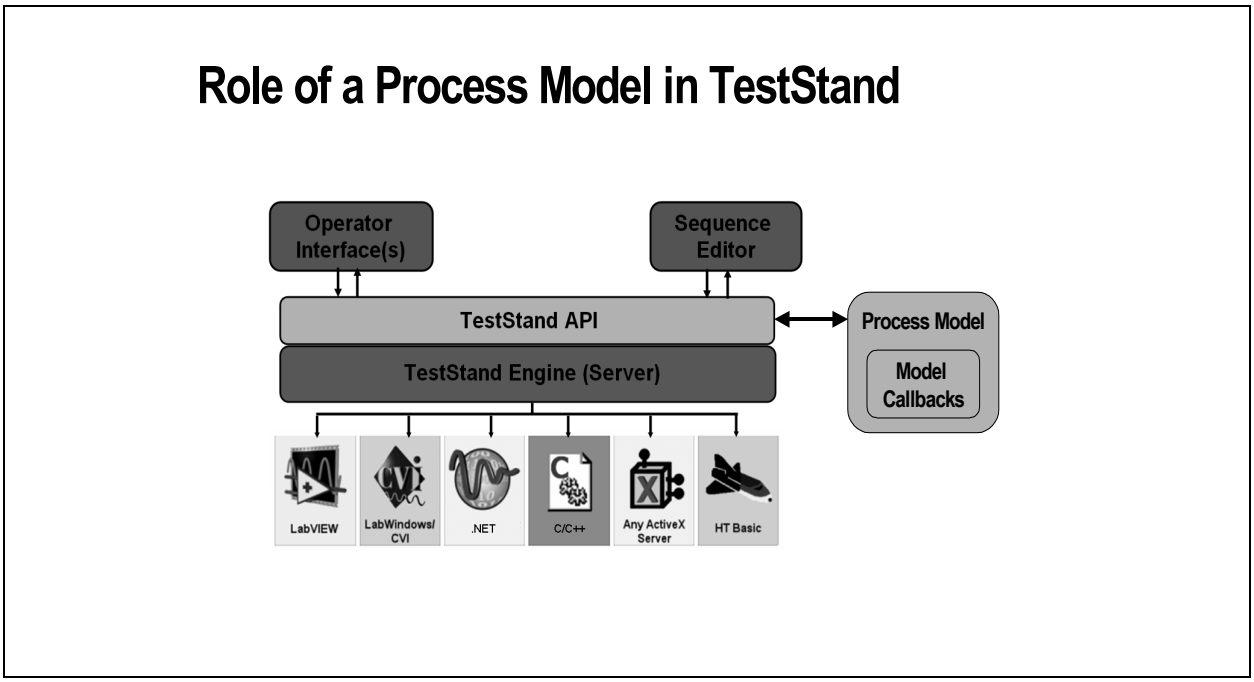
Refer to the *TestStand II: Customization Course Manual* or the *TestStand Reference Manual* for more information about using the Batch and Parallel process models.



Default Process Model: Test UUTs Mode

The flowchart above depicts a simplified version of the order in which the `MainSequence` and all the TestStand callback sequences execute for the Test UUTs mode of the Sequential process model. When the Test UUTs mode initiates, TestStand executes the PreUUTLoop Callback (test system setup operations), followed by the PreUUT Callback (UUT setup operations). These callbacks are always part of the process model, unless you modify the process model yourself. Refer to the *TestStand II: Customization Course Manual* and the *TestStand Reference Manual* for more information about process model customization and callbacks.

The next step is the `MainSequence` step, which you create using the sequence editor. TestStand then executes the PostUUT Callback (UUT clean up operations) and updates a test report. Execution then returns to the PreUUT Callback for the next UUT. On completion of testing, TestStand steps out of the loop, generates a test report, and executes the PostUUTLoop Callback test for system clean up operations.



Role of a Process Model in TestStand

The figure above shows the role of a process model and callbacks in the TestStand architecture. The TestStand Engine executes an entry point of a process model sequence file to direct the flow of operations in the executing test sequence. During process model execution, the engine initiates callbacks at the appropriate time to perform their predetermined tasks. In this way, you can implement modifications without directly modifying the TestStand Engine. Refer to the *TestStand II: Customization Course Manual* and *TestStand Reference Manual* for more information about modifications and customizations.

Exercise 2-3: Understanding the Sequential Process Model

Objective: To become familiar with the TestStand Sequential process model by executing a sequence using Execution entry points.

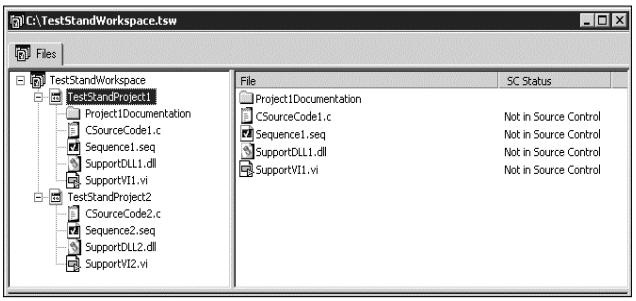
Estimated Time: 20 minutes

Refer to page 2-34 for instructions for this exercise.

Workspaces

Organize sequence files, code modules, and other files into projects.

- Projects enable you to access related files quickly.



Workspaces are the basis for TestStand Distribution!

Workspaces

A workspace is a TestStand file that you can use to organize sequence files, code modules, and other files into projects. Projects enable you to access related files quickly. You can check files and projects in and out of a source code control system from within the TestStand environment. You also can share workspaces with other users by checking a workspace file into source code control.

The Workspace window displays the contents of a TestStand workspace (`.tsw`) file. A workspace file displays a list of any number of TestStand project (`.tsp`) files. In TestStand, a project file can contain any type of file or folder, but a workspace file can contain only projects.

Workspaces are the basis for creating a distribution kit to take your TestStand test system to a target machine. In order to make distribution easier, make sure to keep all pertinent code modules and sequences in a TestStand project and store the projects within a workspace.

Refer to Lesson 12, *Distribution*, of this course manual or to the *TestStand Reference Manual* for more information about distributing TestStand sequences.

Source Code Control

- **Source Code Control—Checks files and projects in and out of a source code control system**
- **Works with any source code control system that supports the Microsoft SCC interface**
  - Microsoft Visual SourceSafe, Perforce, MKS Source Integrity, Rational ClearCase, and Merant PVCS

Source Code Control

TestStand integrates with any source code control (SCC) system that supports the Microsoft SCC interface. You can check files and projects in and out of a source code control system from a TestStand workspace. TestStand has been tested with the following SCC providers: Microsoft Visual SourceSafe, Perforce, MKS Source Integrity, Rational ClearCase, and Merant PVCS.

Refer to the *TestStand Help* for more information about using SCC tools with TestStand.

Sequence File Tools

- **Find**—Search for a specific string, expression, or variable in one or more sequence files.
- **Differ**—Graphical tool that enables you to compare and merge differences between two sequence files.

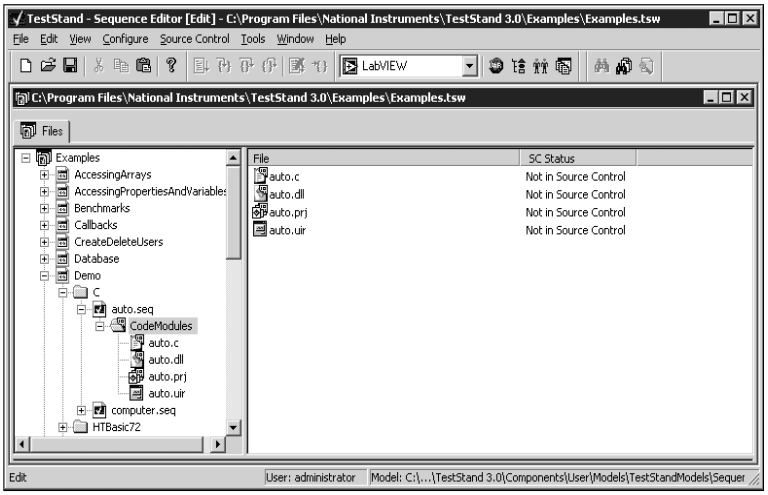
Sequence File Tools

Use the TestStand Find dialog box to search through sequence files for a specific string, expression, or variable. The Find dialog box allows you to specify which sequences should be included in the search. TestStand finds instances of the string or expression and displays them in a dialog box. Click an entry in the dialog box to launch the exact location where TestStand found the string, expression, or variable.

Use the TestStand Differ window in the sequence editor to compare and merge differences between two sequence files. The TestStand Differ window compares the sequence files and presents the differences in a new window.

Refer to the *TestStand Help* for more information about the Find dialog box and the Differ window.

Workspaces and Projects Demo



Workspaces and Projects Demo

The workspaces and projects demo illustrates many of the features discussed on the previous slide. To use this demo, select **File»Open Workspace File** from the sequence editor menu. Open the `Examples.tsw` workspace file located in the `<TestStand>\Examples` directory. This workspace contains projects consisting of all the shipping examples that install with TestStand. Use the tree view to browse through various projects. Notice that the projects can contain sequence files and the associated code modules. You can right-click the Examples node at the top of the tree view to add and remove files from the workspace, as well as check files in and out of source control. The computers used for this course do not have source code control software installed on them, so you will not be able to use the source code control tools within the class.

You can use the TestStand Find tool by opening a sequence file in the sequence editor and selecting **Edit»Find**. TestStand launches a dialog box that allows you to enter a value to search for within a particular sequence file. To search within multiple sequence files, select **Edit»Find in Files**. This more advanced tool allows you to filter the elements and objects that TestStand returns from your query.

You can use the TestStand Differ tool by opening a sequence file in the sequence editor and selecting **Edit»Diff Sequence File With**. Select a second sequence file with which to compare the currently loaded sequence file. TestStand displays a graphical window that shows the differences between the compared sequence files.

Lesson 2 Summary: TestStand Environment

- A test sequence is a series of code modules and flow control settings used to test a UUT
- A sequence has three step groups: Setup, Main, and Cleanup
- Sequence files contain a Main sequence, user-defined sequences, and callback sequences required for sequence execution
- A process model defines the high-level flow of execution for a test executive
- Model callbacks allow the user to customize the process model behavior for each client sequence file that uses the model
- The TestStand default process model is the Sequential model
- Use workspaces, projects, source code control, Find, and Differ tools to manage your test files and structure

Summary

This lesson described the basic components of the TestStand environment: sequences, step groups, and process models.

You can think of TestStand as providing fully customizable "templates" for every aspect of testing. There is a default template available for the process itself, `SequentialModel.seq`, and a default order for the sequence step groups: Setup, Main, and Cleanup. These templates should work for the majority of test environments. As you work through the exercises in this course, you will see that you can easily modify each "template" to suit your particular needs.

From a top down perspective, the process model consists of a particular ordering of sequences and callbacks which match a testing process. The sequences themselves consist of a group of steps. The groups are made up of steps that you configure. These steps correspond to a wide variety of actions, including testing a UUT, message popup for status, and so on. The next lesson discusses how you configure these individual steps to create sequences.

Exercise 2-1 Understanding the Sequence File

Objective: To understand subsequences, setup and cleanup routines, and execution tracing.

Subsequences allow you to modularize logical portions of a test into smaller sequences that are easier to manage and maintain. Breaking up a sequence file into manageable segments is done for the same reason a programmer breaks up sections of a program into many separate functions. Each smaller segment has a very specific purpose and contributes some basic function to the calling sequence it is a part of. In addition to the sequence it was designed for, many other sequences can use this subsequence. Proper use of subsequences improves the reusability and modularity of a test system.

Part A: Calling Subsequences

The following example shows how one sequence can call another sequence. In this example, the Main Sequence calls two subsequences. All three sequences are contained within a single sequence file.

1. Select **Start»Programs»National Instruments»TestStand 3.5» Sequence Editor** to open the sequence editor.
2. Select **File»Open** in the sequence editor and open the `Ex 2-1 Computer Motherboard Test.seq` file located in the `C:\Exercises\TestStand I` directory. Figure 2-1 shows the computer motherboard test sequence file.

   Additional Information
   This sequence file includes the CPU Test and CPU Diagnostics subsequences. In TestStand, a sequence can call another sequence in the same sequence file or in a different sequence file. In this case, the subsequences are in the same sequence file. The light blue icon for the step type denotes a call to a subsequence. Refer to Lesson 3, *Creating Sequences*, for more information about sequences.

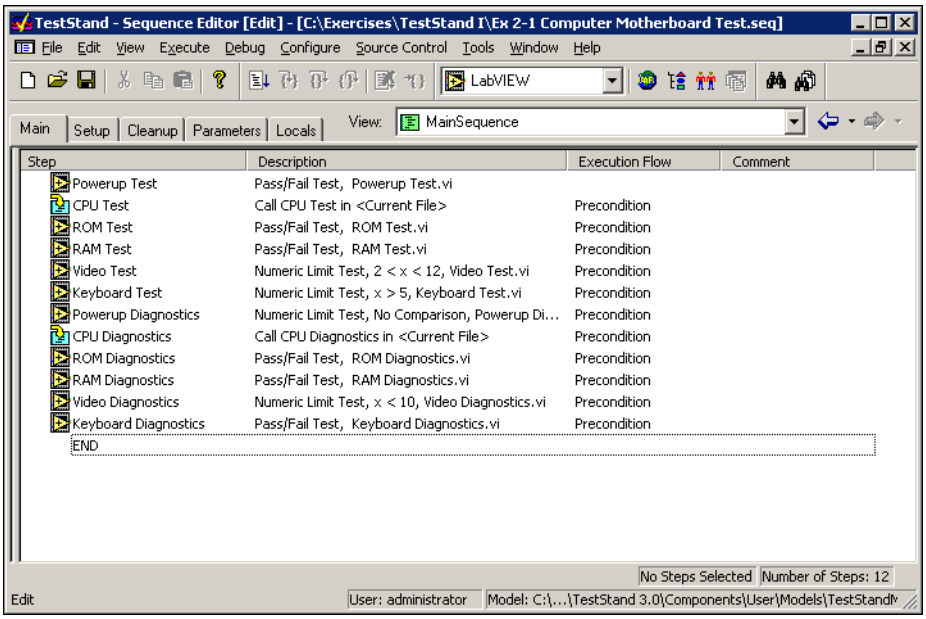


Figure 2-1. Computer Motherboard Test Sequence File

3. Select **Execute»Single Pass** to execute the sequence.
4. Do not select any tests to fail when prompted.
5. Observe the test report when it displays.

   Additional Information
   In a report, the step results of subsequences are separated from the step results of the calling sequence.
6. Select **File»Close** or click the **X** icon in the window title bar to close the Execution window.

7. To open a subsequence, right-click the blue sequence call icon, shown at left, and select **Open Sequence** from the context menu, as shown in Figure 2-2. The selected sequence call displays.

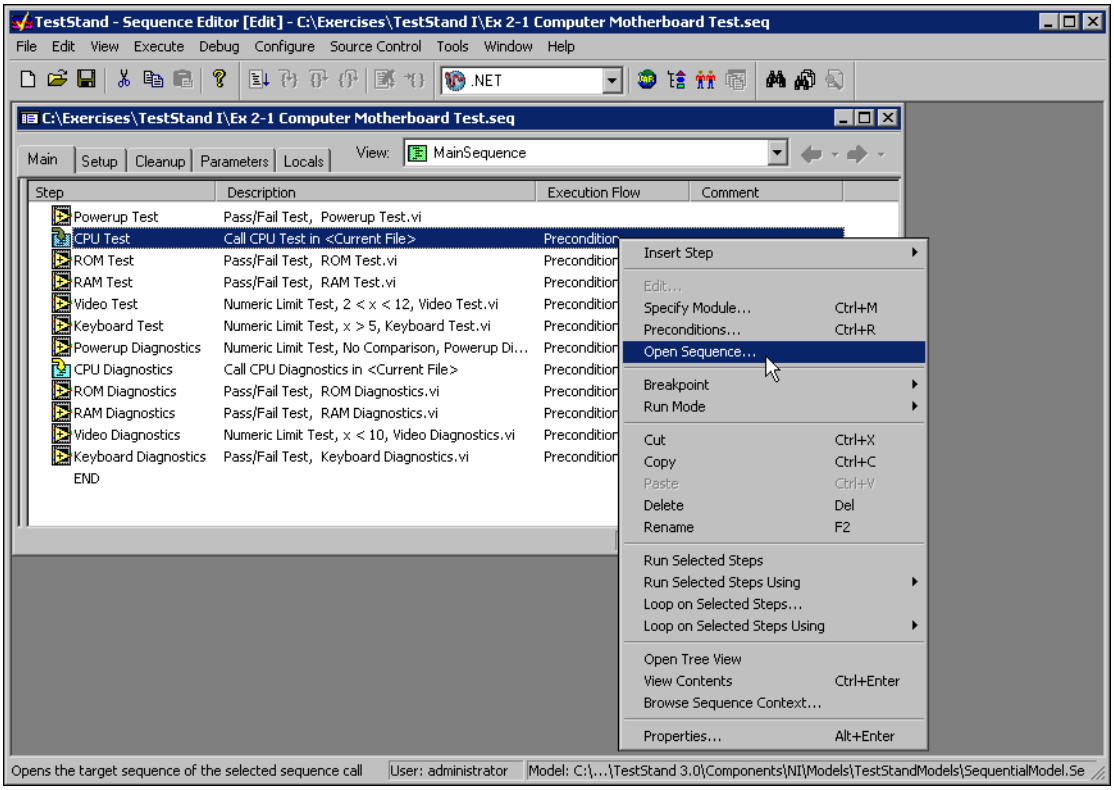


Figure 2-2. Opening a Subsequence

Additional Information

The sequence editor and operator interfaces allow several sequence files to open simultaneously. To toggle between open sequences in different sequence files, use the Window menu. You can also use the browser-style arrows to navigate back and forth among sequences and subsequences.

In this exercise, both sequences are located in the same file; therefore, toggling between sequences files is unnecessary. To toggle between sequences in a single file use the **View** ring control or arrow buttons.

8. Click the back arrow button, shown at left, on the toolbar and return to the previously viewed Sequence File window.

Optional: Using Setup and Cleanup Groups

The window for an individual sequence has five tabs: Main, Setup, Cleanup, Parameters, and Locals. Click a tab to select which part of the sequence to view.

The Main, Setup, and Cleanup tabs each show one step group in the sequence. To view the contents of each tab, click the tab. The steps you insert in each step group have the following purposes:

- **Main**—Test the UUT.
- **Setup**—Initialize or configure instruments, fixtures, and the UUT.
- **Cleanup**—Power down or reset instruments, fixtures, and the UUT.

The Main step group contains the bulk of the steps in a sequence, typically the steps test the UUT. In this sequence, the Main step group contains the steps that test the UUT, such as the Powerup Test, ROM Test, and RAM Test.

The Setup step group contains steps that initialize or configure the test system. In this sequence, the Setup step group contains the Simulation Dialog step that launches the dialog box used to select which computer components should fail. It also contains a step to turn on a simulated vacuum table for testing each UUT.

The Cleanup step group contains steps that power down or deinitialize the instruments, fixtures, and UUTs. In this sequence, the Cleanup step group contains a step for powering down a simulated vacuum table that was turned on in the Setup step group.

Refer to Lesson 4, *TestStand Parameters, Variables, and Expressions*, for more information about the Parameter and Local tabs.

Part B: Tracing Executions

1. Select **Execute** from the sequence editor menu and make sure the Tracing Enabled option is enabled, indicated by a checkmark next to the **Tracing Enabled** menu item. If no checkmark is present, select **Execute»Tracing Enabled** to enable execution tracing. A yellow pointer icon should follow the steps as they execute when you enable execution tracing.
2. To slow the execution tracing, select **Configure»Station Options** to open the Station Options dialog box, shown in Figure 2-3.

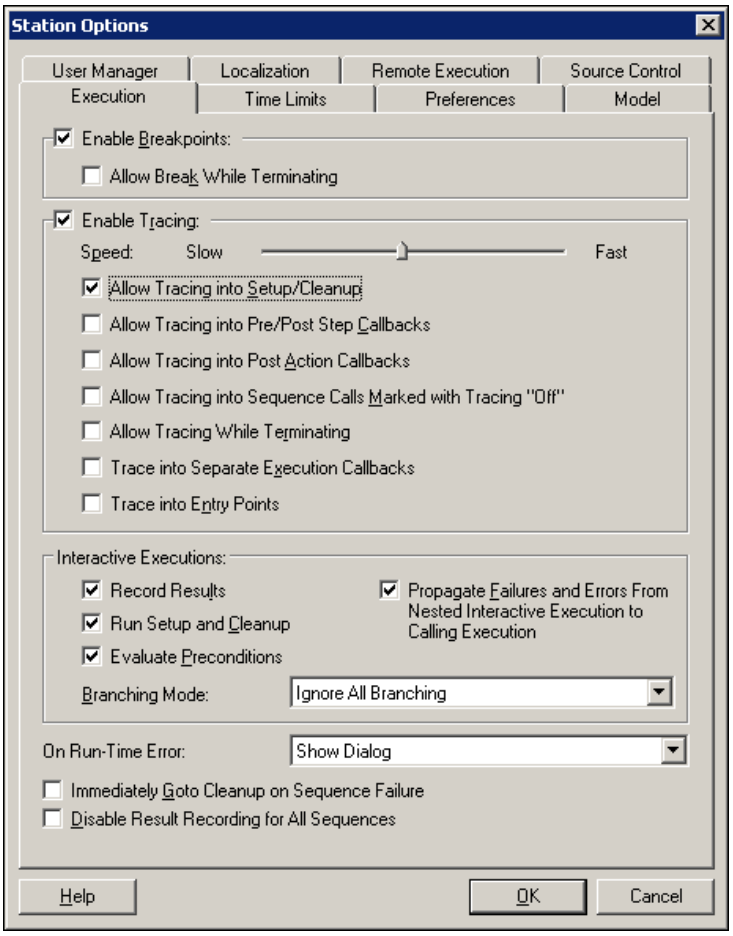


Figure 2-3. Station Options Dialog Box

3. Adjust the execution tracing speed in the Station Options dialog box.
   a. Move the **Speed** slider control toward **Slow** to introduce a delay during tracing.
   b. Enable the **Allow Tracing into Setup/Cleanup** option.
   c. Click **OK** to close the Station Options dialog box.
4. Make sure the Sequence File window for the `Ex 2-1 Computer Motherboard Test.seq` file is the active window. Select **Execute» Single Pass** to start another execution and notice how the tracing affects the view of the execution.
5. When the execution completes, close the Sequence File and Execution windows.

End of Exercise 2-1

Exercise 2-2 Running a Sequence File with Breakpoints and Single Stepping

Objective: To become familiar with using breakpoints and debugging tools in the sequence editor.

When you create a sequence file, you might want to observe sequence execution to make sure the sequence behaves as you need. During debugging, you might want to observe execution in some sections of a sequence but not others. In this exercise, you want TestStand to execute the sequence until it reaches a section you want to analyze.

TestStand allows you to insert convenient stopping points, known as breakpoints, in the sequence. Like most modern development systems, TestStand allows you to set and remove multiple breakpoints for every sequence file.

Part A: The Step Debugging Tools

1. In the sequence editor, select **File»Open** and open the `Ex 2-2 Computer.seq` file located in the `C:\Exercises\TestStand I` directory.
2. Click the space to the left of the `CPU Test` to set a breakpoint on that test. A red stop icon appears, as shown in Figure 2-1, to indicate the breakpoint.

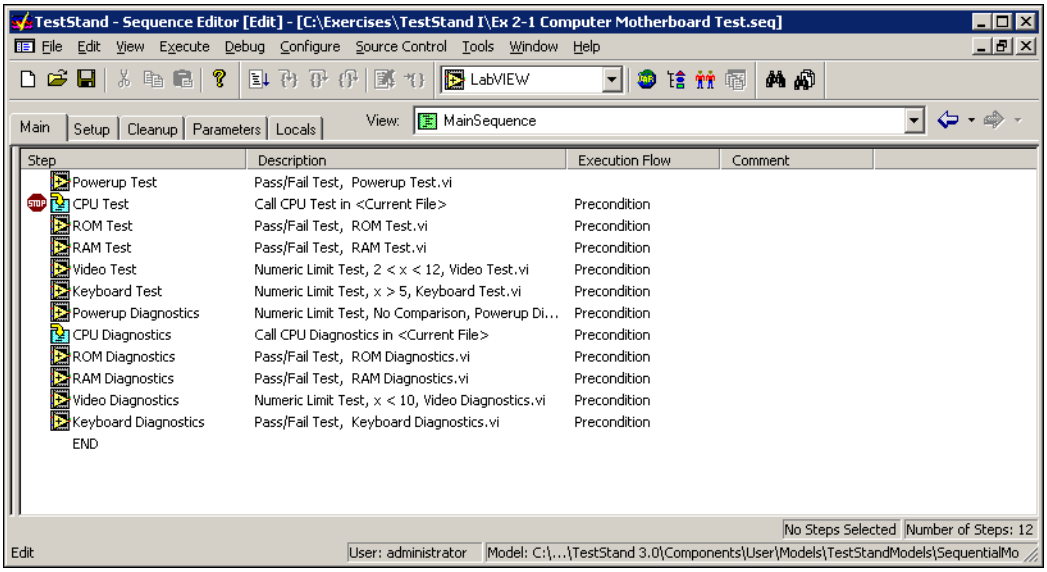


Figure 2-1. CPU Test Breakpoint

Tip Click the stop icon again, to remove a breakpoint.

3. Click **Run**, shown at left, on the sequence editor toolbar. This executes the sequence using the Test UUTs Execution entry point.
4. Enter any number into the UUT Information dialog box and click **OK**.
5. Do not select any tests to fail when prompted.

   Notice that the execution pauses at the first breakpoint, in this case, the CPU Test, because you inserted a breakpoint at this step. The step into, step over, and step out debugging tools should be enabled on the toolbar.

6. Click **Step Into**, shown at left, to step into the CPU Test subsequence and pause at the first step.

7. Click **Step Over**, shown at left, to move to the next step.

8. Click **Resume** to continue execution. While in debug mode, the Run button changes to the Resume button.

9. Before the execution completes, click **Terminate Execution**, shown at left, to terminate the execution and display the results of all steps executed in the report.

Tip If your execution reaches the UUT Result dialog before you have a chance to terminate, start a new UUT and then terminate the execution,

10. Click **Terminate** in the Testing Terminated for Current UUT dialog box that launches.
11. Close the report and return to the Sequence File window.
12. Right-click the breakpoint to open the Breakpoint Settings dialog box.
13. To configure the breakpoint to only be active when Locals.PowerFail is `True`, enter `Locals.PowerFail` in the **Condition** text box as shown in Figure 2-2. This configuration will cause execution to break on the CPU Test step when the Powerup Test step fails.

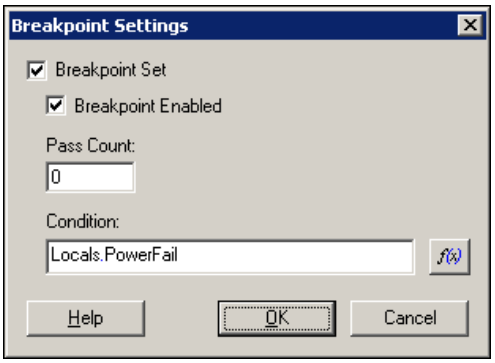


Figure 2-2. Breakpoint Settings Dialog Box

14. Click **OK** to close the Breakpoint Settings dialog box. Notice that the breakpoint icon appears bright red, indicating that it is a conditional breakpoint.
15. Click **Run** to run the sequence again. Enter any number in the UUT Information dialog box. Do not select any tests to fail. Note that the sequence executes without breaking.
16. Enter another number in the UUT Information dialog box. Select the Power test to fail in the Test Simulator dialog box. The sequence breaks at the CPU Test step and the debugging tools activate.
17. Click **Resume** to continue execution.
18. Click **Stop** in the UUT Information dialog box to end the execution. Close the report and return to the Sequence File window.

Note The operator interface also contains the debugging tools described in this section. You can locate the debugging tools in the menus of the operator interface.

End of Part A

Part B: Using the Watch Expression Pane

1. Click the stop icon to the left of the `CPU Test` step to remove the breakpoint.
2. Set a new breakpoint at the `Powerup Test` step by clicking to the left of the step.
3. Click **Run** to execute the sequence. Enter any number into the UUT Information dialog box.
4. Do not select any tests to fail in the Test Simulator dialog box. Click **OK** in the Test Simulator dialog box to execute the sequence.

   The Watch Expression pane in the lower right corner of the Execution window displays the values of variables and expressions during sequence execution to make sure that the values are being set as expected. The sequence editor updates the values in the Watch Expression pane when execution suspends at a breakpoint.

Note You can only use the Watch Expression pane while the execution is running. Therefore, you should place a breakpoint in a sequence file before it begins running if there are variables or expressions you want to watch while debugging the sequence file.

5. To add a new value to the the Watch Expression pane, right-click the pane and select **Add** from the context menu, as shown in Figure 2-3.

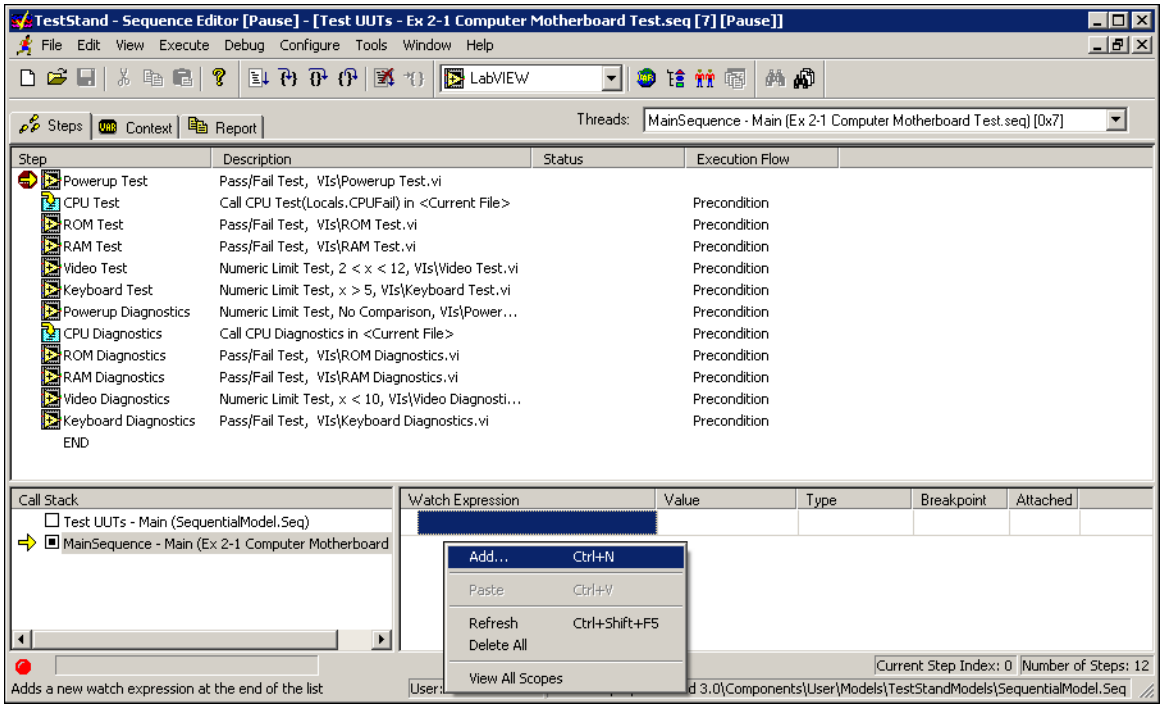


Figure 2-3. Adding a Watch Variable to the Watch Expression Pane

6. In the Watch Expression Settings dialog box that opens, you can select which variables or expressions to watch. You also can configure watch expressions to be active for the current execution only, for a particular sequence, or for an entire sequence file. The settings remain when you open and close the sequence file. Figure 2-4 shows the Watch Expression Settings dialog box.

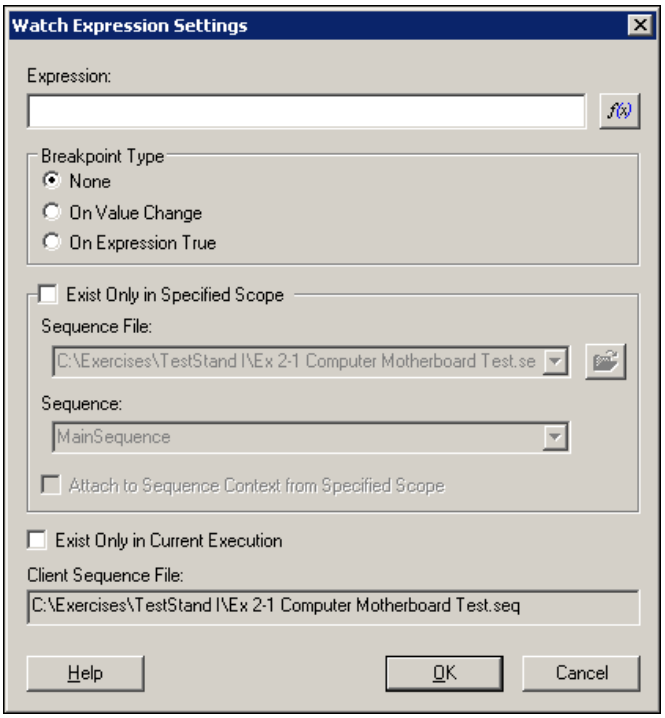


Figure 2-4. Watch Expression Settings Dialog Box

7. Click **Expression Browse**, shown at left, in the Watch Expression Settings dialog box to launch the Expression Browser dialog box.
8. In the Expression Browser dialog box, click the **+** sign next to `Locals` to expand the Locals container.
9. Select the variable, `PowerFail`, and click **Insert**.
10. Click **OK** to close the Expression Browser dialog box.
11. Click OK to close the Watch Expression Settings dialog box and insert the `PowerFail` variable into the Watch Expression pane. Figure 2-5 shows the Watch Expression pane with the inserted variable.

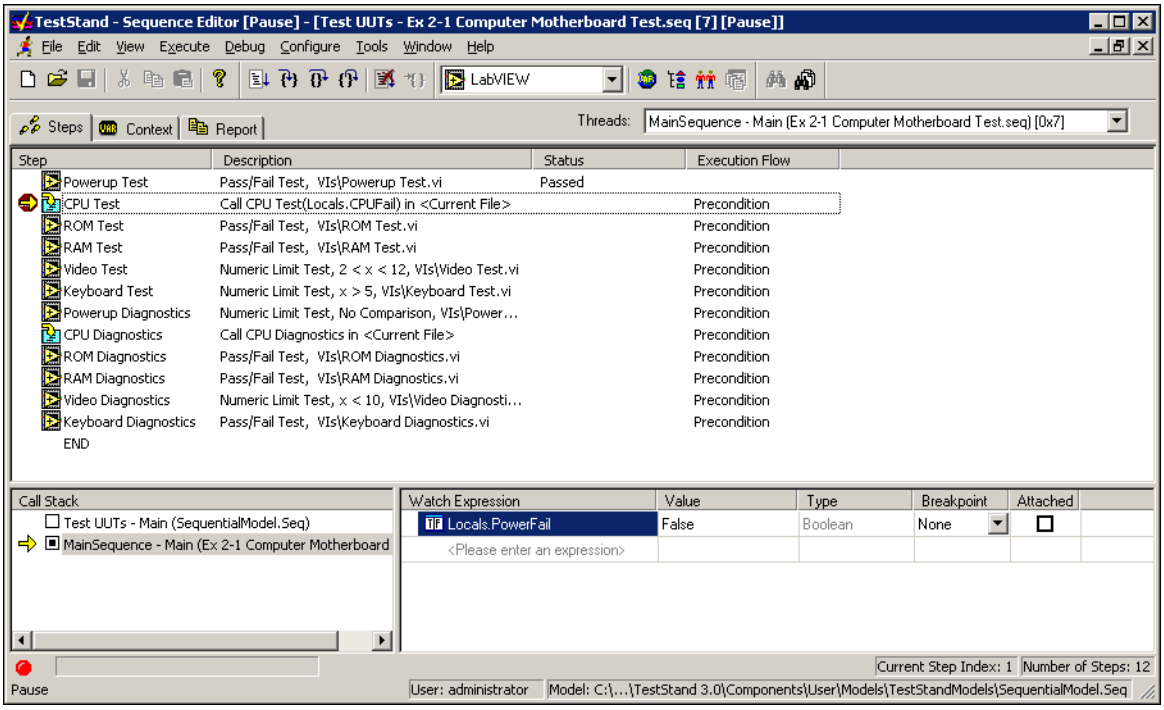


Figure 2-5. Watch Expression Pane with PowerFail Variable Added

12. Repeat steps 5 through 11 to insert the `CPUFail`, `ROMFail`, `RAMFail`, `KeyboardValue`, and `VideoValue` variables. Figure 2-6 shows the completed Watch Expression pane.

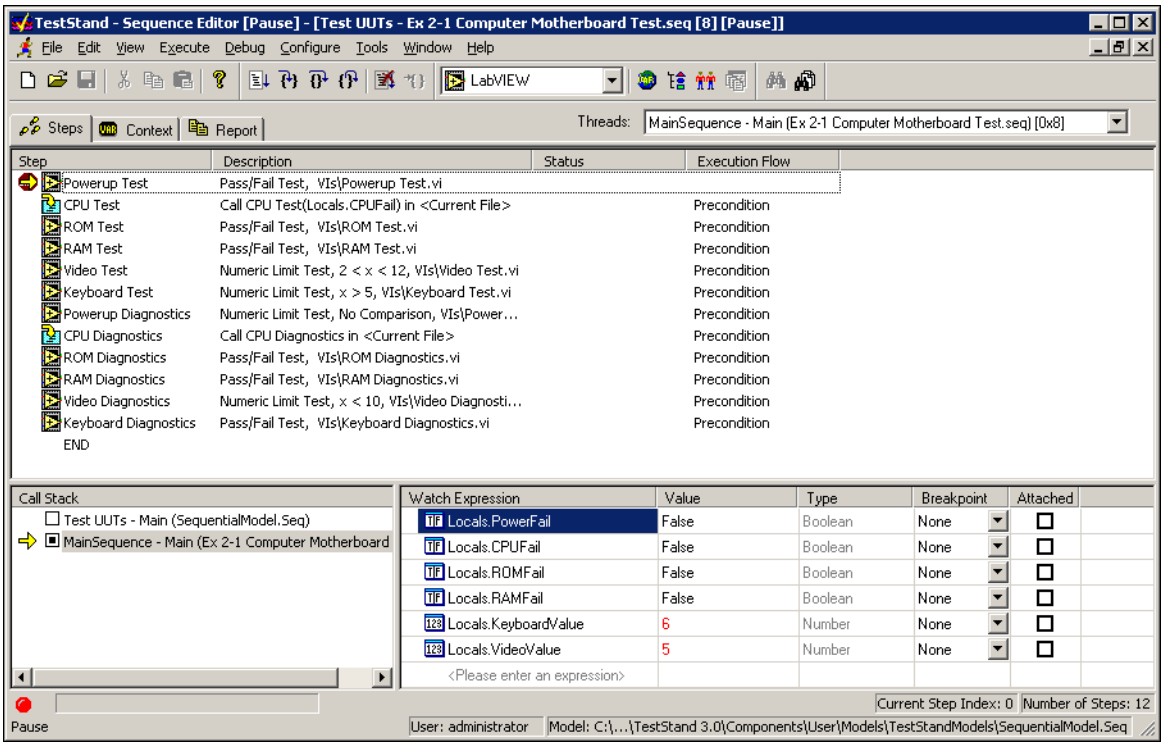


Figure 2-6. Completed Watch Expression Window

13. Click **Resume** to continue the execution of this sequence.
14. When prompted, enter another number in the UUT Information dialog box.
15. Select the Power test to fail in the Test Simulator dialog box.
16. Click **OK** to continue the sequence execution and observe the Watch Expression pane.

    The value for PowerFail changes to `True`. Notice that after you select the Power test to fail and click **OK** in the Test Simulator dialog box, the value of the variable changes.
17. Click **Resume** to complete the execution of the sequence.

    Continue running this sequence, selecting a different test or series of tests to fail. Watch how different variables change in response to the selections you make each time. This is a simple use of the Watch Expression pane, but it demonstrates how you can use this debugging tool to examine variables and expressions during sequence execution.
18. Close all Execution and Sequence File windows. Save changes.

Optional: Adding Values to the Watch Expression Pane

The Expression Browser dialog box is a helpful tool for creating expressions, using functions, and adding values to the Watch Expression pane. However, there is another method to add values to the Watch Expression pane.

1. Select **File»Open** and open the `Ex 2-2 Computer.seq` file located in the `C:\Exercises\TestStand I` directory. Notice that the breakpoint you added is still present in the sequence.
2. Select **Execute»Single Pass** and run the sequence again, with the breakpoint at the PowerUp Test step still in place. Notice that the watch expressions are still present.
3. Select the CPU test to fail when prompted.
4. Click the **Context** tab. Figure 2-7 shows the context view of the Sequence File window.

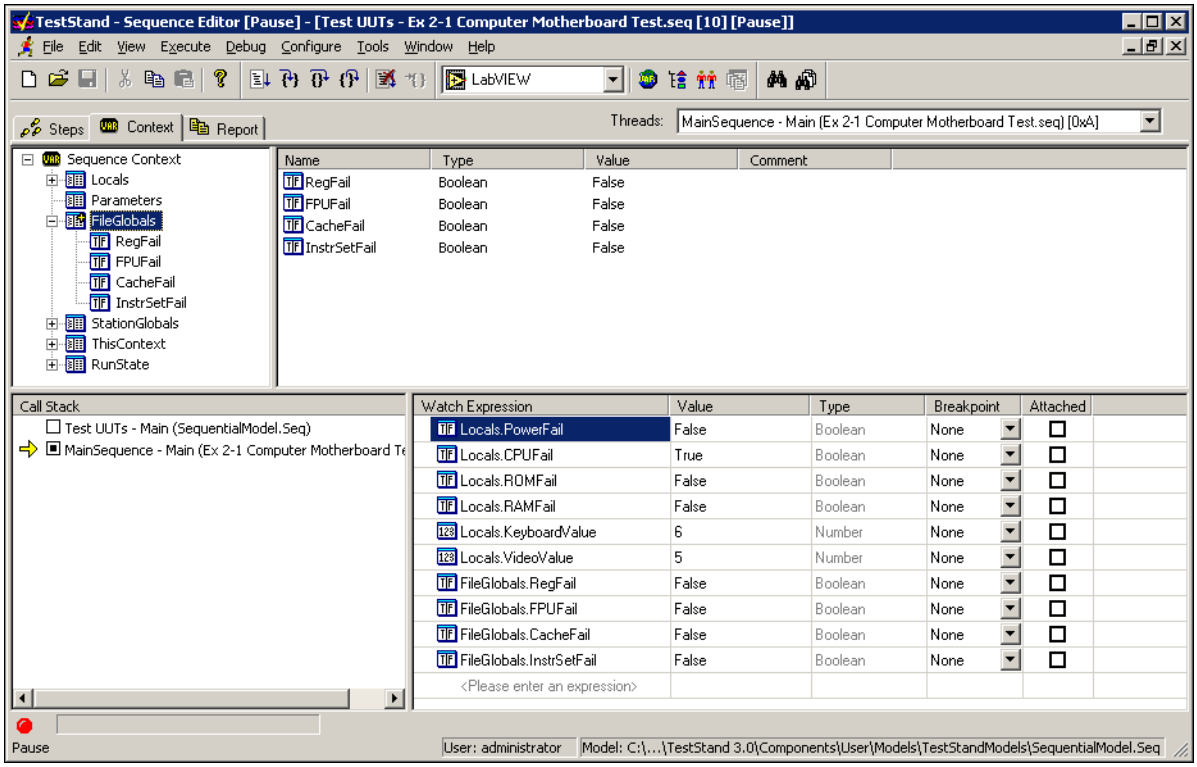


Figure 2-7. Context View

The context view contains the same sequence context information available in the Expression Browser dialog box. However, you can select an expression to watch from the context list and drag it to the Watch Expression pane.

5. Click the **+** sign next to `FileGlobals` to expand the `FileGlobals` container.

6. Using the drag and drop method, add the `RegFail`, `FPUFail`, `CacheFail`, and `InstrSetFail` variables to the Watch Expression pane. You can also select the entire `Locals` container and drag it to the Watch Expression pane.
7. Continue running the test. Observe the values for the expressions.
8. When you finish this exercise, close the Sequence File and Execution windows.

End of Part B

End of Exercise 2-2

Exercise 2-3 Understanding the Sequential Process Model

Objective: To become familiar with the TestStand Sequential process model by executing a sequence using Execution entry points.

Testing a UUT requires more than just executing a set of tests. Typically, the test executive must perform a series of operations before and after it executes the sequence that performs the actual tests. Common operations include identifying the UUT, notifying the operator of pass/fail status, generating a test report, and logging results. These operations define the testing process. This set of operations and their flow of execution is called a process model. Having a process model is essential so you can write different test sequences without repeating standard testing operations in each sequence.

Part A: Running a Sequence Directly

1. Select **File»Open** and open the `Ex 2-3 Computer Motherboard Test.seq` file located in the `C:\Exercises\TestStand I` directory.
2. Select **Execute»Break at First Step** to cause the execution to pause at the first step of the sequence for debugging purposes.
3. Select **Execute»Run MainSequence**. Figure 2-1 shows the execution paused at the first step.

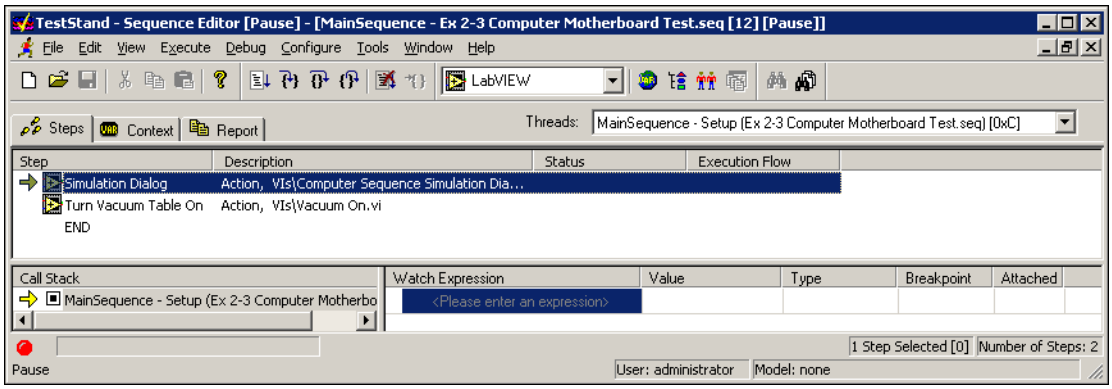


Figure 2-1. Paused Execution with Break at First Step Enabled

4. Click **Step Over** on the toolbar or select **Debug»Step Over** to single step through the sequence. Continue single-stepping through the sequence until the execution completes.

   Notice that only the steps in the `MainSequence` of the `Ex 2-3 Computer Motherboard Test.seq` file executed. No additional steps ran and no report generated.

Part B: Running a Sequence Using the Sequential Process Model

A process model defines a set of entry points. Each entry point gives the test station operator different ways to invoke a Main sequence. For example, the default Sequential process model in TestStand provides two entry points: Test UUTs and Single Pass. The Test UUTs entry point initiates a loop that repeatedly identifies and tests UUTs. The Single Pass entry point tests a single UUT without identifying it. Such entry points are called Execution entry points. Execution entry points appear in the Execute menu of the sequence editor or operator interface.

Complete the following steps to run the `Ex 2-3 Computer Motherboard Test.seq` sequence using the different Execution entry points.

1. Select **Execute»Break at First Step** to remove the checkmark and disable the option.
2. Select **Execute»Single Pass**.
3. Select one of the tests to fail in the Test Simulator dialog box.

   Additional Information
   After TestStand executes the steps in the `MainSequence`, the Single Pass entry point generates a test report. Examine the test report. The report contains information about the results of each step that TestStand executes. Refer to Lesson 7, *Configuring TestStand*, for more information about test reports.
4. Close the Execution window.
5. Select **Execute»Test UUTs**.

   Before executing the steps in the `MainSequence`, the process model launches a UUT Information dialog box requesting a serial number.
6. Enter any number into the UUT Information dialog box and click **OK**.
7. Select a test to fail in the Test Simulator dialog box and observe the Execution window as the sequence executes.

   After completing the steps in the `MainSequence`, the process model launches a dialog box to indicate the sequence result.
8. Click **OK** to close the dialog box. The process model displays the UUT Information dialog box again.
9. Enter another serial number and click **OK** to execute another sequence.

   Repeat this procedure for several serial numbers, then click **Stop** in the UUT Information dialog box to end the execution. TestStand generates a test report for all UUTs. Examine the test report and verify that it has recorded the results for each UUT.
10. Close the Execution window.

Part C: Examining the Sequential Process Model

Process models take the form of sequence files. You can edit the process models like any other sequence. TestStand includes three process models: the Sequential model, the Parallel model, and the Batch model. This exercise examines the Sequential process model.

1. Select **File»Open** and open the `SequentialModel.seq` file located in the `<TestStand>\Components\NI\Models\TestStandModels` directory. This is the default TestStand process model sequence file.

Tip Whenever an exercise provides a path beginning with `<TestStand>` it indicates that the path is part of the files which are installed with TestStand. For example, if you are using TestStand 3.5 and you installed it to the default directory, the path in this exercise would be `C:\Program Files\National Instruments\TestStand 3.5\Components\NI\Models\TestStandModels.`

2. Select **All Sequences** from the **View** ring control in the Sequence File window. Figure 2-2 shows the All Sequences view of the `SequentialModel.seq` sequence file.

   Additional Information

   The `SequentialModel.seq` sequence file has several types of sequences, including Execution entry points, Configuration entry points, and callback sequences. This exercise examines the Execution entry points for the `SequentialModel.seq` sequence file.

   Notice the first two sequences are Test UUTs and Single Pass. A red icon next to these sequences indicates that they are Execution entry points. Multiple Execution entry points enables the operator to invoke a sequence execution in different ways.

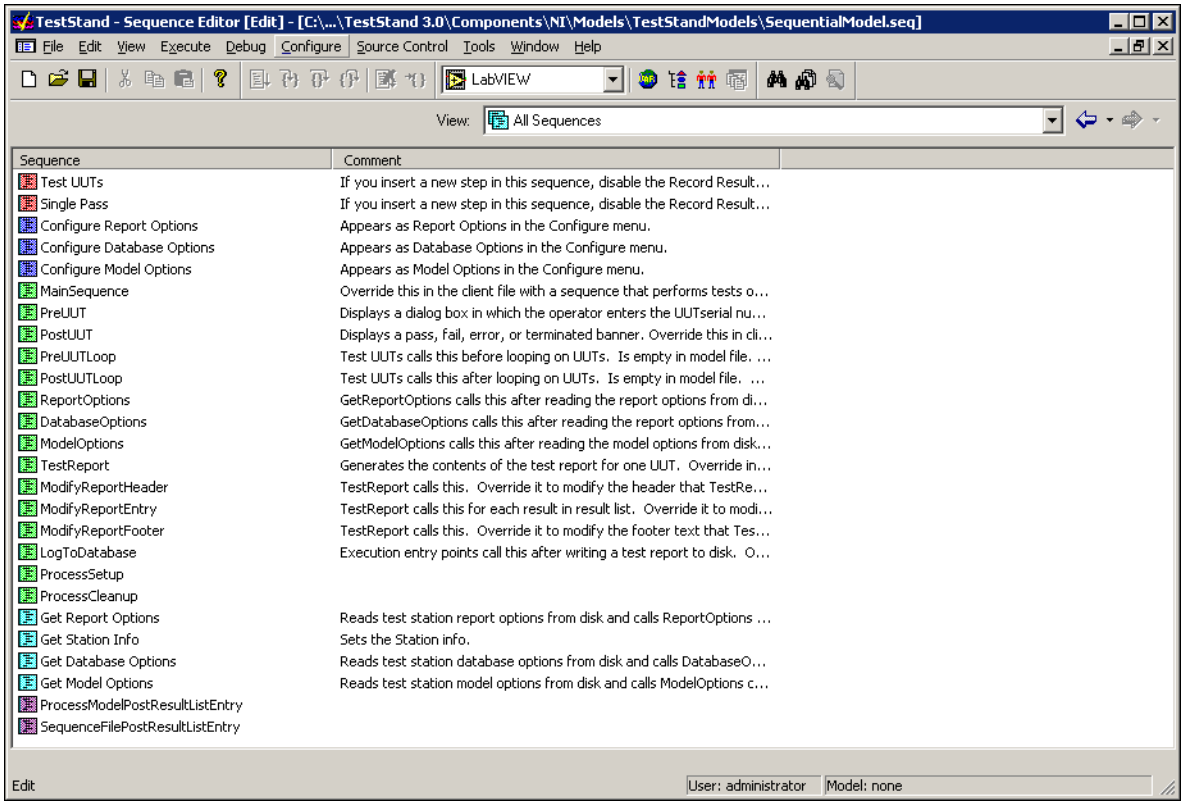


Figure 2-2. All Sequences View of SequentialModel.seq

3. Open the Single Pass sequence by selecting it from the **View** ring control.

   Notice that after some preliminary configuration steps, this sequence executes the MainSequence Callback. This is where the process model executes the test sequence you create in the sequence editor. After calling the MainSequence, the process model executes the TestReport Callback, which generates the test report for the sequence.

4. Right-click the first step of Single Pass sequence and select **Breakpoint»Toggle Breakpoint** from the context menu to set a breakpoint. A red stop icon appears to the left of the step to indicate a breakpoint.

5. Return to the `Ex 2-3 Computer Motherboard Test.seq` Sequence File window by clicking it or selecting it from the Window menu.

6. Select **Execute»Single Pass**.

   The Execution window appears and immediately pauses on the `Get Model Options` step of the Single Pass sequence.

7. Click **Step Over** on the toolbar or select **Debug»Step Over**. Continue stepping over until the sequence pauses with the execution pointer on the `MainSequence Callback` step.

8. Click **Step Into** or select **Debug»Step Into** to step into the `MainSequence Callback` step.

   Notice that the execution has proceeded to the steps of the `Ex 2-3 Computer Motherboard Test.seq` sequence file.

9. Click **Step Out** or select **Debug»Step Out** to return to the process model sequence.

   After the test runs, the sequence execution returns to the steps of the Single Pass sequence.

10. Click the breakpoint to remove it from the `SequentialModel.seq` file before continuing with the execution.
11. Click **Resume** on the toolbar to resume normal execution or select **Debug»Resume**.
12. Close the Execution window and all Sequence File windows.

End of Exercise 2-3

Self Review

1. What is a step?

2. What is the difference between a sequence and a sequence file?

3. List the three step groups within a sequence, and explain what they are used for.

4. What is the difference between the Main sequence and the Main step group?

5. What is execution tracing and when would you use it?

6. Give the definition of a process model.

7. Explain how the process model fits into the TestStand architecture.

8. What is a workspace and when would you use it?

9. What is the TestStand Differ tool used for?

10. What is the TestStand Find tool used for?

Notes

3

Creating Sequences

Lesson 3: Creating Sequences

In this lesson, you will:

- Add steps to a TestStand sequence
- Specify a code module for each step
- Configure step properties

Introduction

This lesson describes the steps involved in creating a TestStand sequence. You will learn how to create a new TestStand sequence, specify the code modules for each step, and configure the step properties.

How to Create Test Sequences

- **Use the sequence editor to build test sequences**
- **Build sequences by adding steps for each test or action you want to execute**
- **Use the following four actions to add each step to the sequence:**
  - Specify the adapter
  - Select the step type
  - Specify the code module
  - Configure the step properties

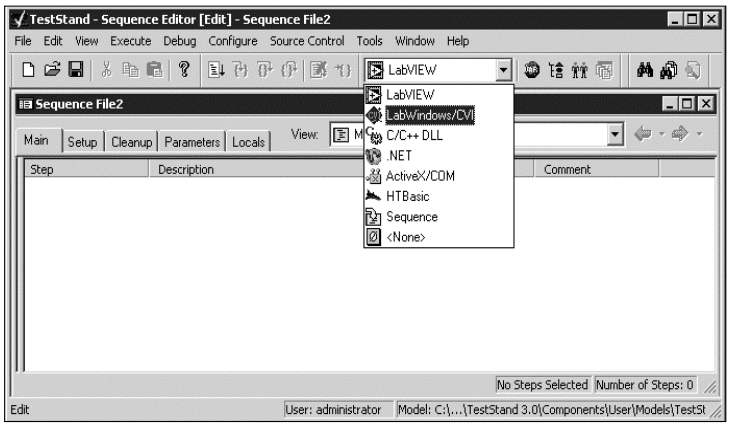
How to Create Test Sequences

The TestStand Sequence Editor is the development environment where you build test sequences. The sequence editor is an intuitive Windows Explorer-style interface you use to copy, move, paste, or create steps to build a sequence. Use the following four steps to create each step:

1. Specify the adapter corresponding to the type of code module that your programming environment produces.
2. Select the step type.
3. Specify the code module the step will call.
4. Configure the various step properties.

Adding a Step to a Sequence

Step 1: Specify the Adapter



Step 1: Specify the Module Adapter

Before you insert a new step, select the appropriate module adapter from the Adapter ring control, as shown above. When invoking code in a code module, TestStand must know the type of code module, how to call it, and how to pass parameters to it.

TestStand includes the following module adapters:

- **LabVIEW Adapter**—Calls LabVIEW VIs with a variety of connector panes.
- **LabWindows/CVI Adapter**—Calls C functions with a variety of parameter types. The functions can be in object files, library files, or DLLs. They also can be in source files that are in the project you are currently using in LabWindows/CVI.
- **C/C++ DLL Adapter**—Calls functions or methods in a DLL with a variety of parameter types, including National Instruments Measurement Studio classes.
- **.NET Adapter**—Calls methods and accesses the properties of objects in a .NET assembly.
- **ActiveX/COM Adapter**—Calls methods and accesses the properties of object in an ActiveX server.
- **HTBasic Adapter**—Calls HTBasic subroutines.
- **Sequence Adapter**—Calls other TestStand sequences with parameters.
- **<None>**—Creates a step that does not call a code module.

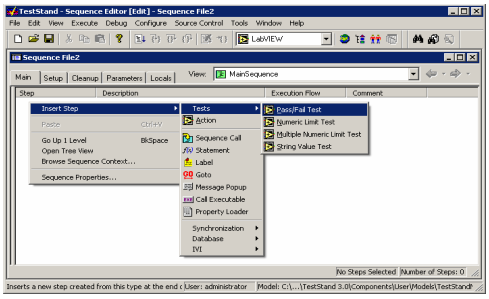
Note You can configure which adapters appear in the Adapter ring control by selecting **Configure»Adapters**.

Adding a Step to a Sequence

Step 2: Select the step type

Built-in step types:

- **Tests**
  - Pass/Fail Test
  - Numeric Limit Test
  - Multiple Numeric Limit Test
  - String Value Test
- **Action**
- **Sequence Call**
- **Statement**
- **Message Popup**
- **Call Executable**
- **Synchronization**
- **Flow Control**
- **Property Loader**
- **Database**
- **Goto**
- **Label**
- **IVI**



Step 2: Select the Step Type

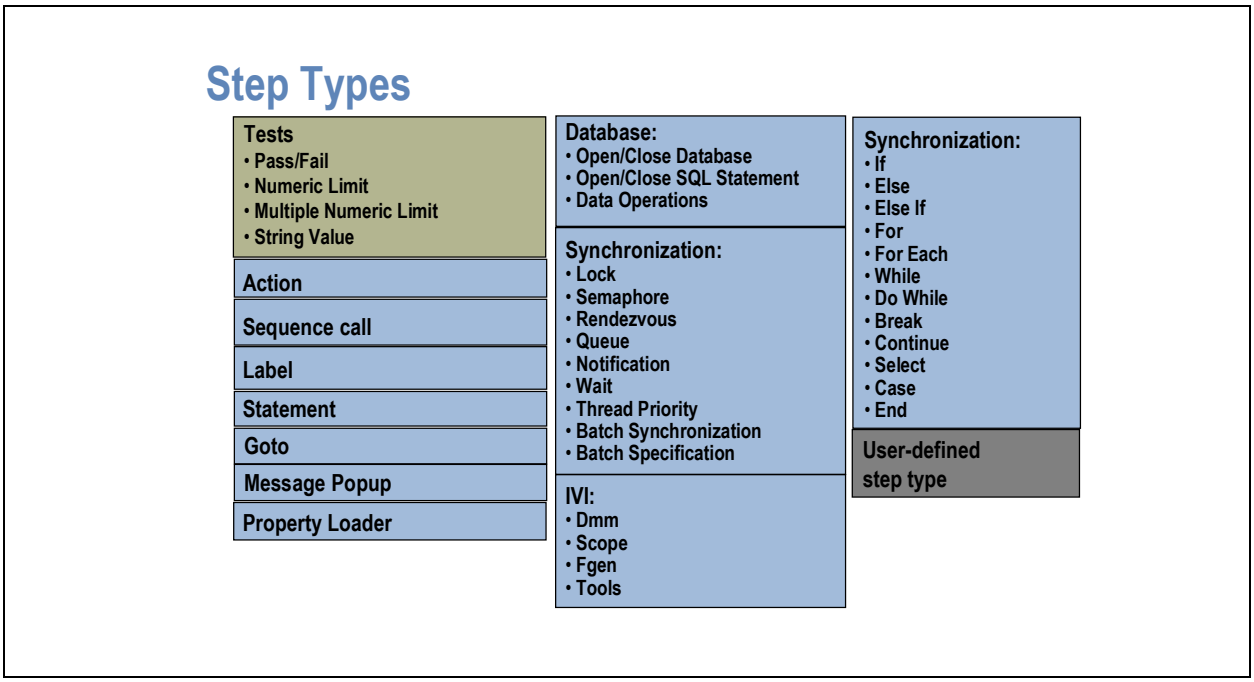
After you select an adapter, right-click in the Sequence File window. Select **Insert Step** from the context menu. Each step type defines the step properties and functionality specific to a task. Additionally, some features are common among all step types.

TestStand includes over 30 built-in step types. You can customize these step types or create your own custom step types. The context menu displays all defined step types, both built-in and user-defined.

TestStand allows you to copy step types and modify their properties to suit your requirements. Refer to Lesson 9, *TestStand Types*, of this manual for more information about modifying and creating custom step types.

The following built-in test step types, available by right-clicking and selecting **Insert Step»Tests**, make pass/fail determinations. Each of these types has different associated data and properties.

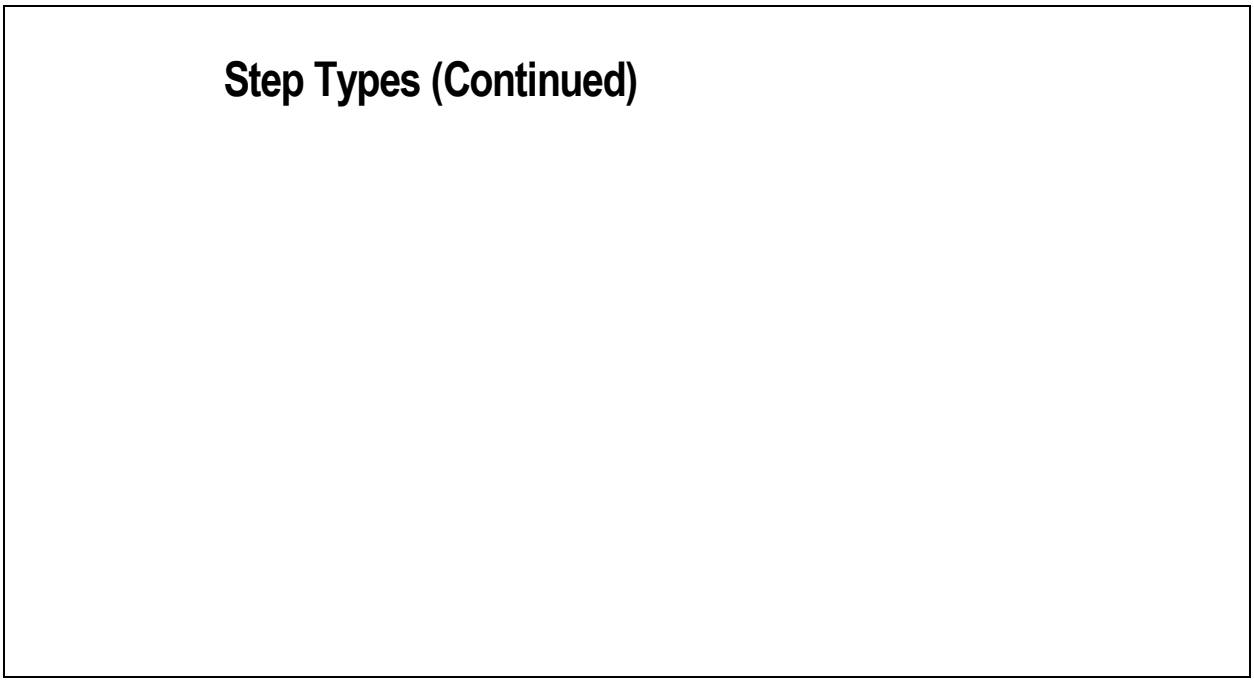
- **Pass/Fail Test**—Passes a Boolean value (`True` or `False`) to determine if the step passes or fails.
- **Numeric Limit Test**—Passes a single numeric value, which is compared to predefined limits of the step.
- **Multiple Numeric Limit Test**—Passes multiple numeric values, which are compared to sets of predefined limits for the step.
- **String Value Test**—Passes a string, which is compared to the expected string for the step.



Step Types

TestStand also includes the following built-in non-test step types:

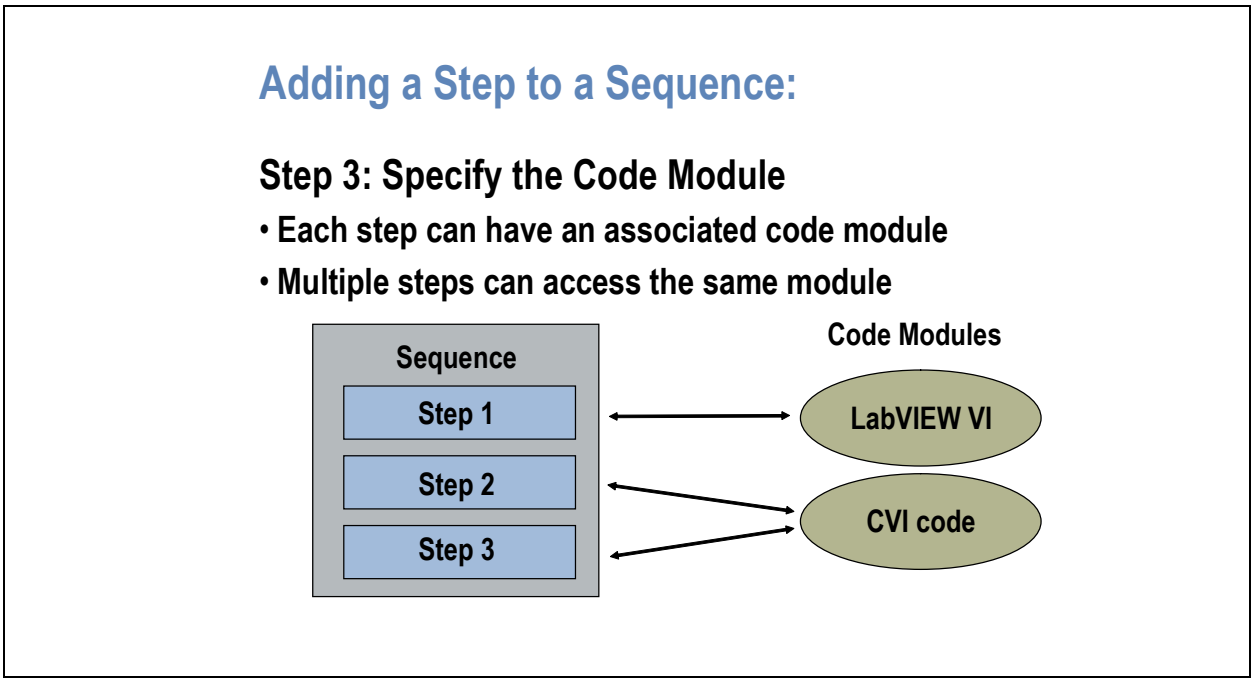
- **Action**—Used to call code modules that do not perform tests but perform actions necessary for testing, such as initializing an instrument. By default, Action steps do not pass or fail. An Action step type executes an operation successfully or generates a run-time error.
- **Sequence Call**—Calls another sequence in the current sequence file or in another sequence file.
- **Statement**—Executes expressions. For example, you can use a Statement step to increment the value of a local variable in a sequence. By default, Statement steps do not pass or fail. If a Statement step cannot evaluate the expression it generates a run-time error.
- **Label**—Used as a target for a Goto step.
- **Goto**—Used to set the next step that the TestStand Engine executes. You usually use a Label step as the target of a Goto step so you can rearrange or delete other steps in a sequence without changing the specification of targets in Goto steps.
- **Message Popup**—Displays messages to the operator and receives response strings from the operator.
- **Call Executable**—Launches an application or runs a system command.



Step Types (Continued)

Property Loader—Dynamically loads the values for properties and variables from a text file, Microsoft Excel file, or a DBMS database at run time. Refer to Lesson 6, *Creating Sequences*, for more information about the Property Loader step type.

- **Synchronization**—Enables you to synchronize, pass data between, and perform other operations in multiple threads of an execution or multiple running executions in the same process.
- **Database**—Enables you to communicate with a database.
- **Interchangeable Virtual Instruments (IVI)**—Enables you to configure and acquire data from IVI class-compliant instruments.



Step 3: Specify the Code Module

After you add a new step to a sequence, you must specify the code module that the step executes. Although some step types, such as Statement or Message Popup, do not require you to specify a code module, many others, such as the Numeric Limit Test do. A code module can be any of the following types:

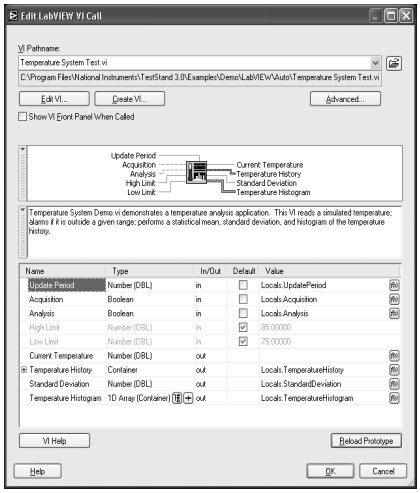
- LabVIEW VI
- LabWindows/CVI source file
- Object file
- Static library
- HTBasic subroutine
- C DLL
- Automation server
- .NET Assembly
- TestStand sequence

Although each step can call only one code module, multiple steps can call the same code module.

A step is a component of a TestStand sequence. A code module is a software module written in an external application development environment (ADE) that performs a particular task.

Specifying the Module

- For a step that uses an Adapter, right-click the step and select Specify Module from the context menu to display the Edit Call dialog box
- TestStand has different dialog boxes for each adapter



Edit LabVIEW VI Call Dialog Box

Specifying the Module

Each module adapter has its own Specify Module dialog box because different information is needed to create the link depending on the programming language used to write the code module. For example, a LabVIEW step needs to know the location of the VI it calls, while a DLL test needs the DLL location, the specific function to call within the DLL, and the parameters to pass to the function.

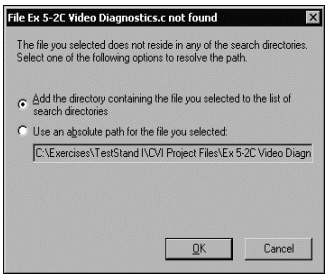
To specify a code module for a step, complete the following steps:

1. Right-click the step and select **Specify Module** from the context menu.
2. In the dialog box that launches, complete the required information for the adapter.

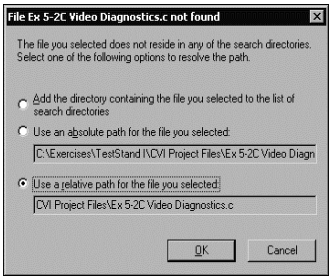
Specifying the Module: Search Directories

Dialog boxes used for resolving module paths

The module is not within a subdirectory of the configured search directories



The third option shows a path that is relative to one of the configured search directories



Specifying the Module: Search Directories

When you select a module and it is not located in one of the configured search directories, one of two dialog boxes launches to prompt you to resolve the path to the module.

If the path to the module is not in a subdirectory of a configured search directory, then you are given two options: add the directory of the module to the list of search directories or reference the module in an absolute path.

If the module is located in a subdirectory of a configured search directory but the Search Subdirectories option was not enabled for that directory, you are given the additional option to reference the module using a path relative to the search directory.

To edit the search directory settings manually, select **Configure»Search Directories**. Notice that when you add a directory to the list of search directories, you have the option of including its subdirectories in the search.

Adding a Step to a Sequence:

Step 4: Configure the Step Properties

- **Each step in the sequence has a set of properties**
- **The following categories are properties associated with each step:**
  - General (includes preconditions)
  - Run Options
  - Post Actions
  - Loop Options
  - Synchronization
  - Expressions
  - Switching

Step 4: Configure the Step Properties

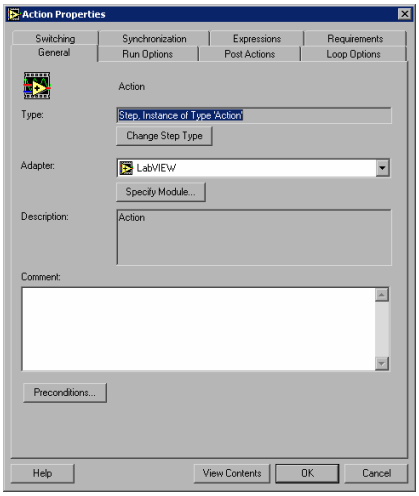
Each step in a sequence contains properties. The final action in adding a step to a sequence is to configure these properties so that the step executes as you want in the sequence. The type of a step determines the set of properties that a set has. All steps have a common set of properties that determine the following attributes: when to load 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, and what conditional actions occur upon step completion.

To access the step properties, right-click the step and select **Properties** from the context menu. The Step Properties dialog box launches.

The Step Properties dialog box includes the following tabs:

- **General**
- **Run Options**
- **Post Actions**
- **Loop Options**
- **Synchronization**
- **Expressions**
- **Switching**

Step Properties: General Tab



Step Properties: General Tab

The General tab of the Step Properties dialog box contains buttons that launch the Specify Module dialog box, the step type-specific Edit dialog boxes, and the Preconditions dialog box. The General tab displays the adapter type, the step type, a description of the step, and user-defined comments.

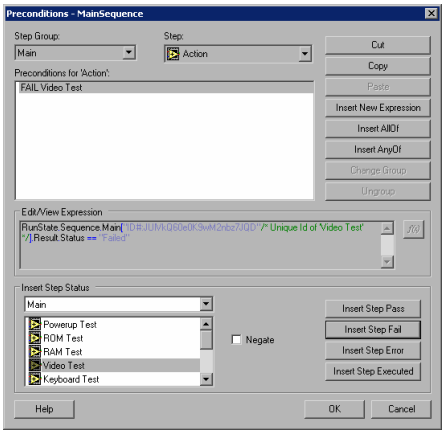
The General tab command buttons vary according to the step type. For example, a Pass/Fail test has a Edit Pass/Fail Source button to modify the data source, while a String Value test has an Edit Expected String button to specify the string for comparison with the step results.

Click the **Specify Module** button to launch the Specify Module dialog box for the selected step. Use this dialog box to specify the code module that the step calls and specify options that TestStand uses when it calls the step. This is the same dialog box that displays when you right-click a step and select **Specify Module** from the context menu. Each adapter has a unique Specify Module dialog box.

You can change the step type using the Change Step Type button. You can change the module adapter by selecting a different module adapter from the ring control. Both of these tools allow you to retain any other settings that are specific to this step.

Step Properties: Preconditions

- **The preconditions property specifies the conditions that must be `True` for a step to execute**
- **You can set sequence execution logic based on the status of other steps, the value of TestStand variables, and the value of user-defined expressions**



Step Properties: Preconditions

Click the **Preconditions** button on the General tab of the Step Properties dialog box to launch the Preconditions dialog box.

For most applications, you want to execute many of the steps in a sequence only under certain conditions. In TestStand, you can set preconditions based on the status of other steps, the value of TestStand variables, or user-defined expressions consisting of one or more of these values. TestStand can also execute different levels of precondition expressions that range from simple to complex, enabling you to create highly flexible sequences.

The main window of the Preconditions dialog box displays the conditions that must be True for the current step to execute. You can organize several precondition statements by using AnyOf or AllOf (Boolean OR and AND statements, respectively).

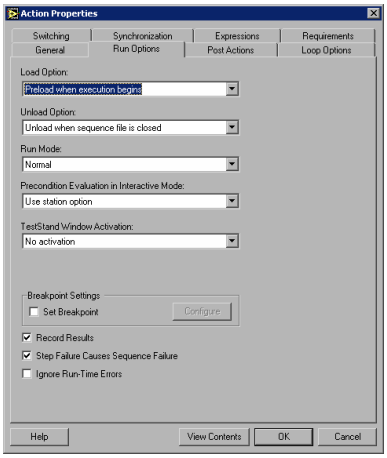
Use the Edit/View Expression section to create your own expressions or to view expressions entered from the Sequence File window.

Use the Insert Step Status section to build precondition expressions for display in the Sequence File window. Refer to the *TestStand Help* for information about precondition expressions.

Step Properties: Run Options Tab

Run Options:

- **Load/Unload dynamic/static code module**
- **Precondition evaluation**
- **TestStand window activation**
- **Record Results**
- **Step Failure Causes Sequence Failure**
- **Debugging features:**
  - Run mode: normal, skip, force pass, force fail
  - Breakpoint Settings
  - Ignore Run-time Errors



Step Properties: Run Options Tab

The Run Options tab allows you to configure how TestStand executes the code module called by the step including Load Option, Unload Option, Run Mode, Precondition Evaluation in Interactive Mode, and TestStand Window Activation. Refer to the *TestStand Help* for information about the options available on the Run Options tab of the Step Properties dialog box.

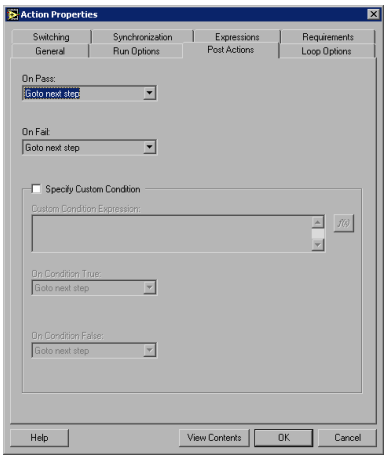
When running in interactive mode, the selected steps in the sequence execute regardless of any branching logic the sequence contains. The selected steps run in the order in which they appear in the sequence.

Note When you use the LabVIEW Adapter and specify the module to show the user interface front panel, TestStand automatically activates the TestStand window when the step completes.

Step Properties: Post Actions Tab

Available Post Actions:

- **Go to a certain step**
- **Call a sequence**
- **Break**
- **Terminate or execute a user-defined expression**

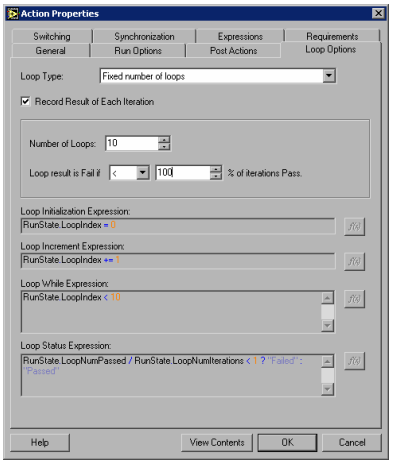


Step Properties: Post Actions Tab

The Post Actions tab in the Step Properties dialog box specifies an action that occurs after a step executes. You can make the action conditional on the Pass/Fail status of the step or on any custom condition. Refer to the *TestStand Help* for information about the options available on the Post Actions tab of the Step Properties dialog box.

Step Properties: Loop Options Tab

Allows looping on a step based on a fixed number of iterations, a pass/fail count, or a user-defined condition.



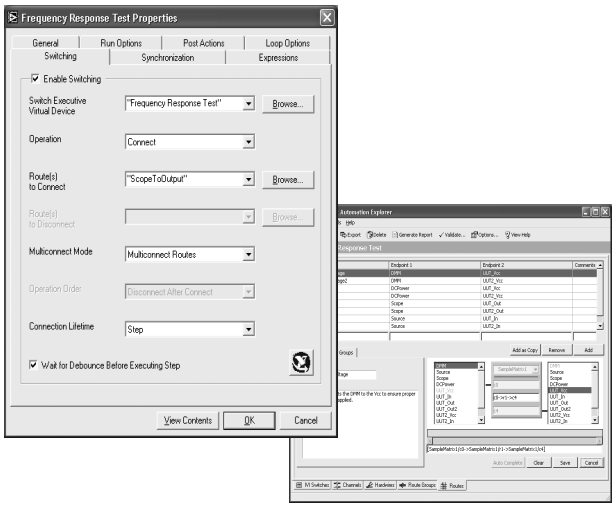
Step Properties: Loop Options Tab

The Loop Options tab in the Step Properties dialog box configures an individual step to run repeatedly in a loop when it executes. You can configure a step to loop for a fixed number of iterations or until a given number of pass or fail counts is achieved. In addition, you can use TestStand expressions to define a custom looping condition.

Refer to Lesson 4, *TestStand Parameters, Variables, and Expressions*, for more information about expressions. Refer to the *TestStand Help* for information about the options available on the Loop Options tab of the Step Properties dialog box.

Step Properties: Switching Tab – NI Switch Executive Demo

Simplify switch programming through integration with NI Switch Executive.



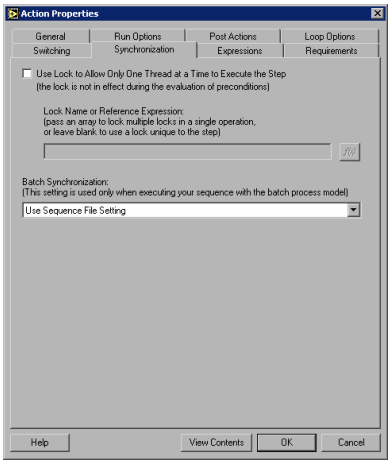
NI Switch Executive Demo

NI Switch Executive is an intelligent switch management and routing application. You can interactively configure and name switch modules, external connections, and signal routes using the graphical end-to-end route editor. The following demo illustrates configuring a TestStand step to automatically perform the switching operations for the step based on the configuration created using NI Switch Executive. Complete the following steps to run the example:

1. Launch NI Measurement Automation Explorer (MAX) and select **Devices and Interfaces**.
2. Select the **SampleSwitchExecutive** virtual switch device. The NI Switch Executive configuration environment is now visible.
3. Browse through the **IVI Switches**, **Channels**, **Hardwires**, **Route Groups**, and **Routes** tabs.
4. Notice the graphical end-to-end route editor on the **Routes** tab.
5. Close NI Switch Executive and return to TestStand.
6. Open the example sequence `SwitchExecutive.seq` located in the `<TestStand>\Examples\Switching` directory and double-click the `Connect and Disconnect w/ Step Lifetime` test step.
7. Click the **Switching** tab in the Step Properties dialog box and notice that TestStand automatically retrieves the available virtual switch device configurations and the corresponding pre-defined routes. TestStand also supports dynamic route creation through NI Switch Executive by using the `FindRoute` function in the `Routes to Connect` expression.

Step Properties: Synchronization Tab

- **Lock—Step must acquire the lock before executing.**
- **Batch Synchronization—Controls the step execution in concurrent batch executions of a sequence.**

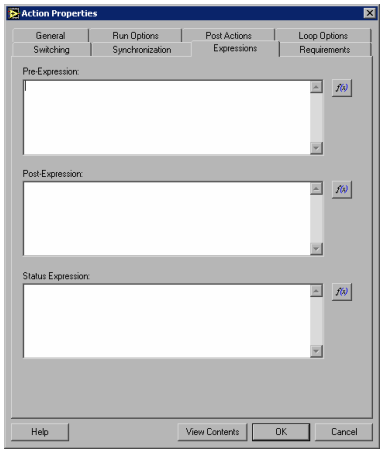


Step Properties: Synchronization Tab

In addition to the Synchronization step type, you can also specify common synchronization properties on any step using the Synchronization tab of the Step Properties dialog box. Refer to the *TestStand Help* for information about the options available on the Synchronization tab of the Step Properties dialog box. Refer to the *TestStand II: Customization Course Manual* for more information about synchronization step types, the Batch process model, and multithreading.

Step Properties: Expressions Tab

Allows you to define arbitrarily complex Pre- and Post-Expressions



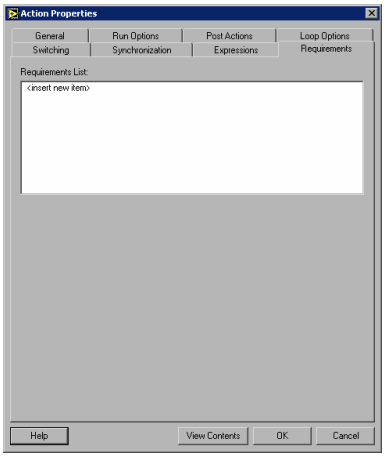
Step Properties: Expressions Tab

Use the Expressions tab of the Step Properties dialog box to specify optional expressions that TestStand evaluates before or after it calls the step module. For example, you can use the Pre-Expression to initialize certain variables required for a particular step, and this can be followed by the Post-Expression, which resets the variables to other values in preparation for other steps. TestStand evaluates the Status Expression to determine whether a step passes or fails.

For a Numeric Limit Test step, the default status expression compares the result returned from the code module with high and low limits. This determine whether the status of the step is `Passed` or `Failed` using the selected comparison criteria.

Step Properties: Requirements Tab

Provides a mechanism for notating product and unit requirements that the step covers.



Step Properties: Requirements Tab

Exercise 3-1: Creating Steps in the Sequence Editor

Objective: To add new steps to the `MainSequence` that set off an alarm if the ROM Test fails.

Estimated Time: 20 minutes

Refer to page 3-26 for instructions for this exercise.

Exercise 3-2: Configuring Loop Options

Objective: To build a sequence that loops on a step depending on the step results.

Estimated Time: 20 minutes

Refer to page 3-36 for instructions for this exercise.

Exercise 3-3: Creating a Sequence

Objective: To create a sequence using different step types.

Estimated Time: 20 minutes

Refer to page 3-41 for instructions for this exercise.

Exercise 3-4: Running Simultaneous Sequences (Optional)

Objective: To show how TestStand can run sequences simultaneously.

Estimated Time: 10 minutes

Refer to page 3-57 for instructions for this exercise.

Lesson 3 Summary: Creating Sequences

- **Use the sequence editor to edit and build sequences**
- **Complete the following four actions to add a step to a sequence:**
  - Specify the adapter
  - Select a step type
  - Specify a code module
  - Configure the step properties

Summary

In this lesson, you learned the steps for building a sequence. When you create a sequence step you specify an adapter, select a step type, specify a code module, and configure the step properties. Remember that there are three step groups in each sequence that you can insert a step into: Setup, Main, and Cleanup.

Your decision for choosing an adapter is based on the type of code module that your programming environment produces. Notice that TestStand provides the flexibility of using different adapters for each step. Through this adapter interface, you can quickly port existing test code to TestStand.

Specifying the code module allows you to select an existing code module, or you can build a new module starting from a code template.

Finally, after adding a step to a sequence, you configure the step's properties. In configuring the step properties, you can specify a wide range of behaviors, such as skipping the step (Run Options), only executing the step under certain conditions (Preconditions), looping on the same step a fixed number of times or until a condition occurs (Loop Options), performing an action after the step completes (Post Action), or initializing or computing a value before or after the step runs (Expressions).

Exercise 3-1 Creating Steps in the Sequence Editor

Objective: To add new steps to the MainSequence that set off an alarm if the ROM Test fails.

The TestStand Sequence Editor is a graphical user interface (GUI) for creating and modifying sequence files. You can use the sequence editor to add steps to the sequence file to create a testing sequence for a UUT. When a test fails, you need a way to notify the operator of the failure. Including failure information in the report might be insufficient, and some other notification method can be more appropriate. In this exercise, you will add a step to the sequence file that sets off an audible alarm when the ROM test fails for any UUT.

1. In the sequence editor, select **File»Open** and open the `Ex 3-1 Add Beep Step.seq` sequence file located in the `C:\Exercises\TestStand I` directory.
2. Select the **C/C++ DLL Adapter** from the **Adapter** ring control, as shown in Figure 3-1.

   When you insert a step that calls a code module, you must specify the module adapter that the step uses. This step uses the C/C++ DLL Adapter to call a function in a DLL.

Figure 3-1. Adapter Ring Control

3. Right-click the `ROM Test` and select **Insert Step»Action** from the context menu.

   Additional Information
   An Action step type calls code modules that perform actions necessary for testing, such as initializing an instrument. By default, Action steps do not pass or fail, so you do not use them for actual product tests. Refer to the *TestStand Help* for more information about steps you use for testing.
4. After you insert the step, the Action step is highlighted. Enter `AlarmStart` and press the **<Enter>** key to rename the step.

Tip To rename a step, you can also right-click the step and select **Rename** from the context menu.

5. Right-click the `AlarmStart` step and select **Properties** from the context menu or double-click the `AlarmStart` step to open the Step Properties dialog box for the step, as shown in Figure 3-2.

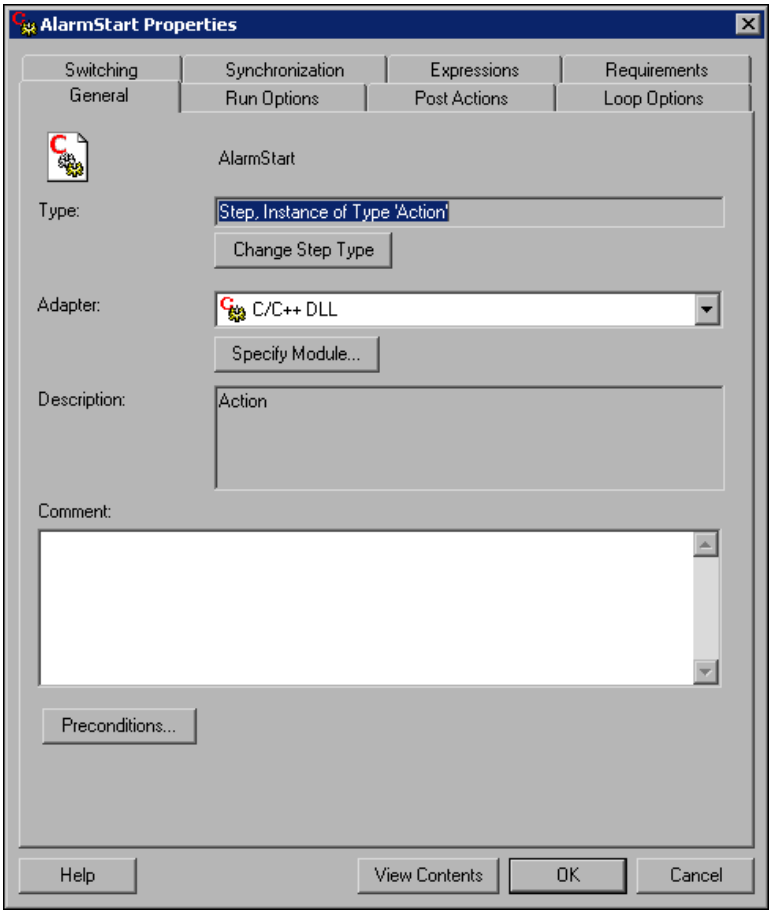


Figure 3-2. Step Properties Dialog Box

Additional Information

The Step Properties dialog box contains following options for configuring the step:

- When to load the step
- What conditions will cause the step to execute
- What information TestStand examines to determine whether a test passes or fails
- Whether TestStand executes the step in a loop
- Ways to synchronize the step with other executions

6. Click **Specify Module** to display the Edit C/C++ DLL Call dialog box.

7. Click **File Browse**, shown at left, to open the Select DLL dialog box, shown in Figure 3-3.

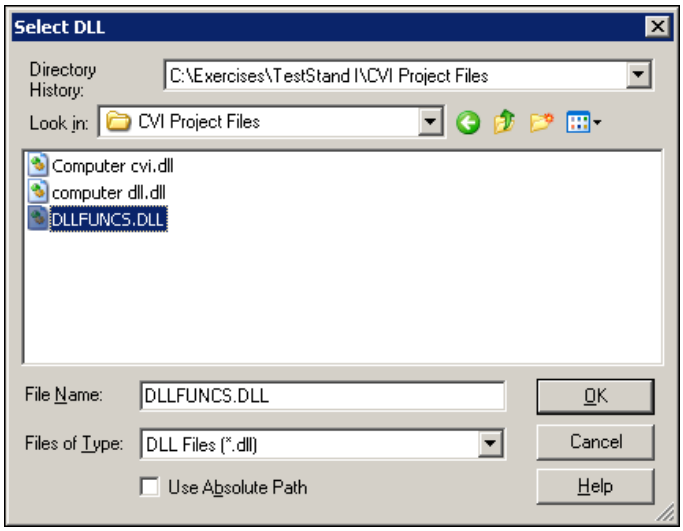


Figure 3-3. Select DLL Dialog Box

8. Select the `DLLFUNCS.DLL` file from the `C:\Exercises\TestStand I\CVI Projects File` directory and click **OK**.

Note If you cannot find this DLL, open Windows Explorer and select **Tools»Folder Options**. Select the **View** tab and enable the **Show Hidden Files and Folders** option.

Figure 3-4 shows a dialog box that might open with options for the file path. If this dialog box appears, select the **Add the directory containing the file you selected to the list of search directories** option to add the path to the search directories TestStand searches.

Additional Information

If the module is not within a subdirectory of the configured search directories, TestStand prompts you to resolve the path by adding the file you selected to the search directories, use an absolute path to the file, or use a relative path for the file.

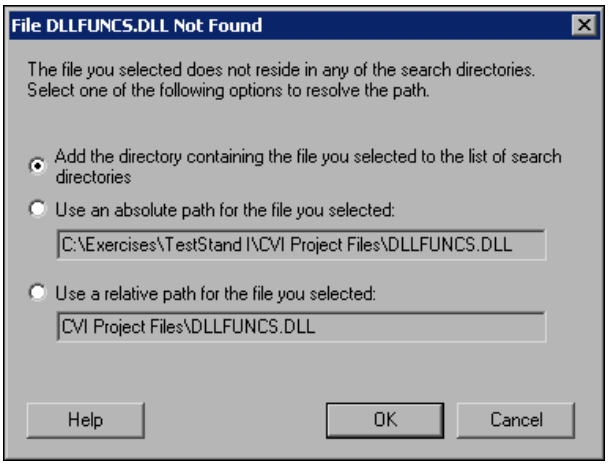


Figure 3-4. File Path Options Dialog Box

9. Click **OK** to return to the Edit C/C++ DLL Call dialog box.
10. Select the **StartBeep** function from the **Function Name** ring control.

    All functions exported from the DLL are listed under the Function Name ring control. This exercise only uses the StartBeep and StopBeep functions.
11. Enable the **Edit Prototype** option and set the parameters as shown in Figure 3-5. The tone parameter sets the beep frequency, which is 1,000 Hz in this case.

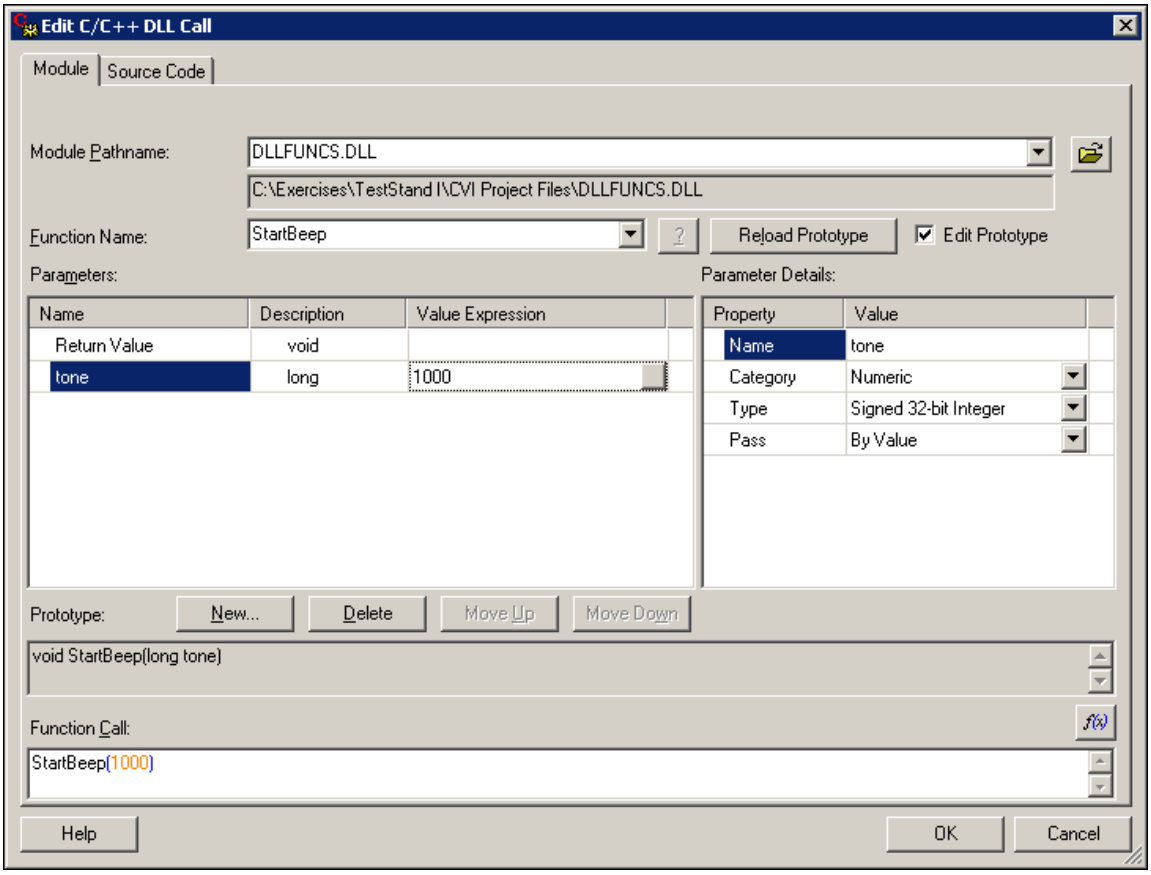


Figure 3-5. Module Tab Parameter Settings

Note You must generally build a DLL with a type library for all of your parameters to appear in the Parameter ring control and for the function prototypes to appear in the Prototype section at the bottom of the Edit C/C++ DLL Call dialog box. To create a type library in LabWindows/CVI, you must create a Function Panel file first.

If you are using LabWindows/CVI 7.1 or later and TestStand 3.1 or later, function information will be embedded within the DLL itself and you do not need to create a type library for the parameters and prototypes to appear.

12. Click **Preconditions** to open the Preconditions dialog box, shown in Figure 3-6.

    Additional Information

    A precondition specifies the conditions that must be `True` for TestStand to execute a step during the normal flow of execution in a sequence. For example, you might want to run a step only if a previous step passes. You enter the conditions that control when a particular test executes. In this case, the AlarmStart step should execute only if the ROM Test step fails.

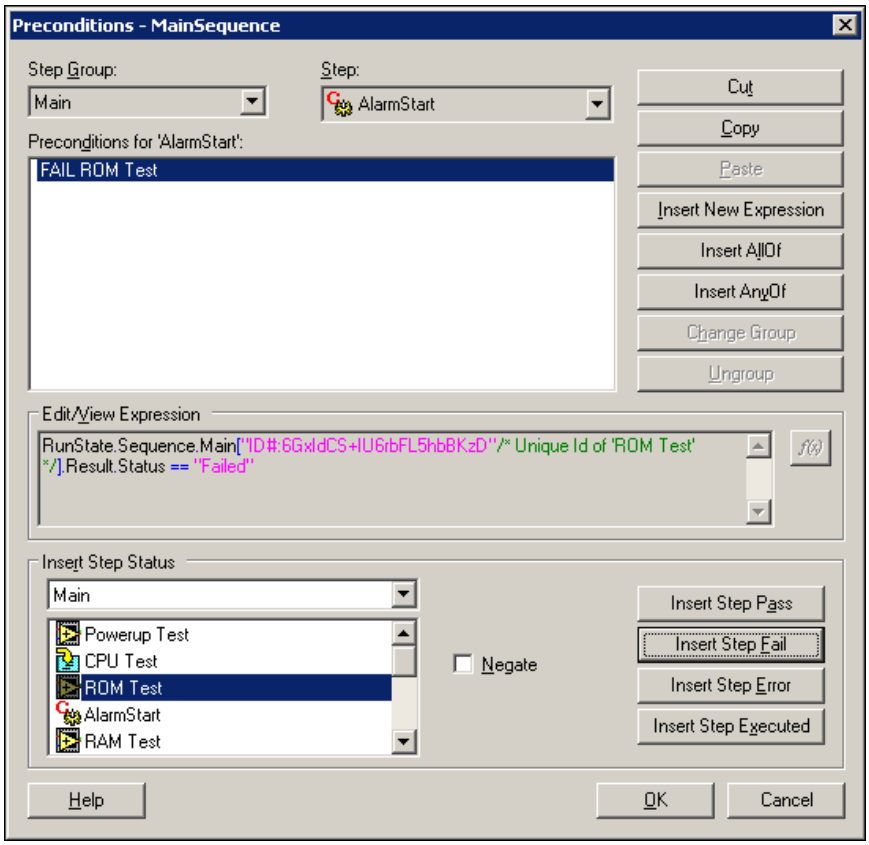


Figure 3-6. Preconditions Dialog Box

13. Select the **ROM Test** in the **Insert Step Status** section and click **Insert Step Fail**.

    Additional Information

    This inserts the condition that the AlarmStart step executes only if the ROM Test step fails. Notice that other options for conditions include running the current step only if another test passed, caused an error, or has executed.

    The Edit/View Expression text box shows the TestStand syntax for the selected precondition. The string of random numbers and letters is actually a unique ID for that step. This allows you to use the same step name more than once in a sequence.

    You can also create expressions that TestStand evaluates and uses as conditions for executing a step. Refer to Lesson 4, *TestStand Parameters, Variables, and Expressions*, for more information about inserting more than one precondition and the use of TestStand expressions.

14. Click **OK** to close the Preconditions dialog box.

Note If the Step Properties dialog box is closed, double-click the `AlarmStart` step in the Sequence File window to open it.

15. Click the **Run Options** tab in the Step Properties dialog box to examine the run options. Ensure that the **Run Mode** ring control is set to **Normal** and the **Record Results** option is enabled.

    Browse through the **Load Option**, **Unload Option**, **Run Mode**, **Precondition Evaluation in Interactive Mode**, and **TestStand Window Activation** options. Check that the settings match those shown in Figure 3-7 before continuing.

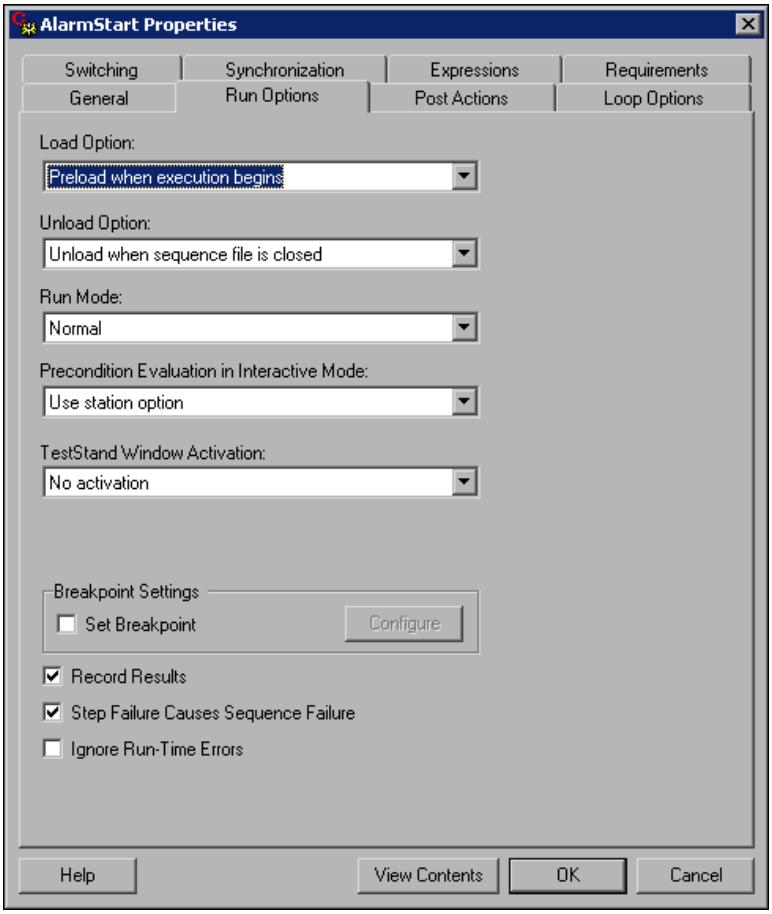


Figure 3-7. Run Options Tab Settings

Additional Information

Although some options are self-explanatory, it is helpful to understand the various run mode options and how to use them to configure TestStand step execution. The options are useful in debugging, testing the various states of each step, and understanding how the preconditions relate to each other. Click **Help** on the Step Properties dialog box for a complete description of all the options on the Run Options tab.

16. Click the **Post Actions** tab in the Step Properties dialog box. Use the Post Actions tab to specify an action to occur after a step executes. Verify that the options are configured as shown in Figure 3-8 before continuing to the next step.

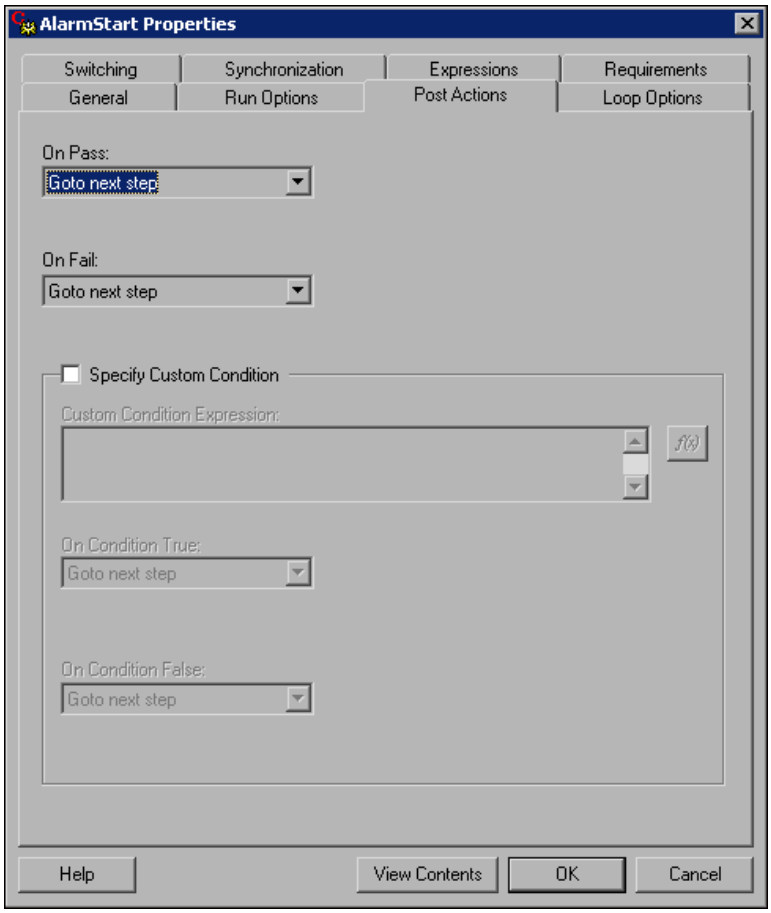


Figure 3-8. Post Actions Tab Settings

Additional Information

A step that is specified as an Action step, such as the StartBeep step, can only use the Specify Custom Condition action because an Action step does not Pass or Fail. For other types of test steps, the post action can be conditional on the pass/fail status of the step or any custom condition expression. For example, you might want to jump to a particular step in the sequence if a step fails. By default, the On Pass and On Fail actions are Goto next step.

17. Click the **Loop Options** tab in the Step Properties dialog box. You can use the Loop Options tab to configure an individual step to run repeatedly in a loop when it executes.
18. Configure the `AlarmStart` step loop options as follows:
    a. Select **Fixed number of loops** from the **Loop Type** ring control.

b. Set the **Number of Loops** to `25`.

This configures the AlarmStart step to run 25 times and return a `FAILED` status if any iteration fails. This is determined by the loop result, which you set to `Fail` if less than 100 percent of the iterations pass. The completed **Loop Options** tab should be similar to Figure 3-9.

Additional Information

As you enter changes in the Loop Options tab, TestStand automatically builds an expression in the Loop While Expression and Loop Status Expression controls. An expression is a formula that TestStand evaluates to obtain a new value from the values of multiple variables or properties. Refer to Lesson 4, *TestStand Parameters, Variables, and Expressions*, for more information about expressions.

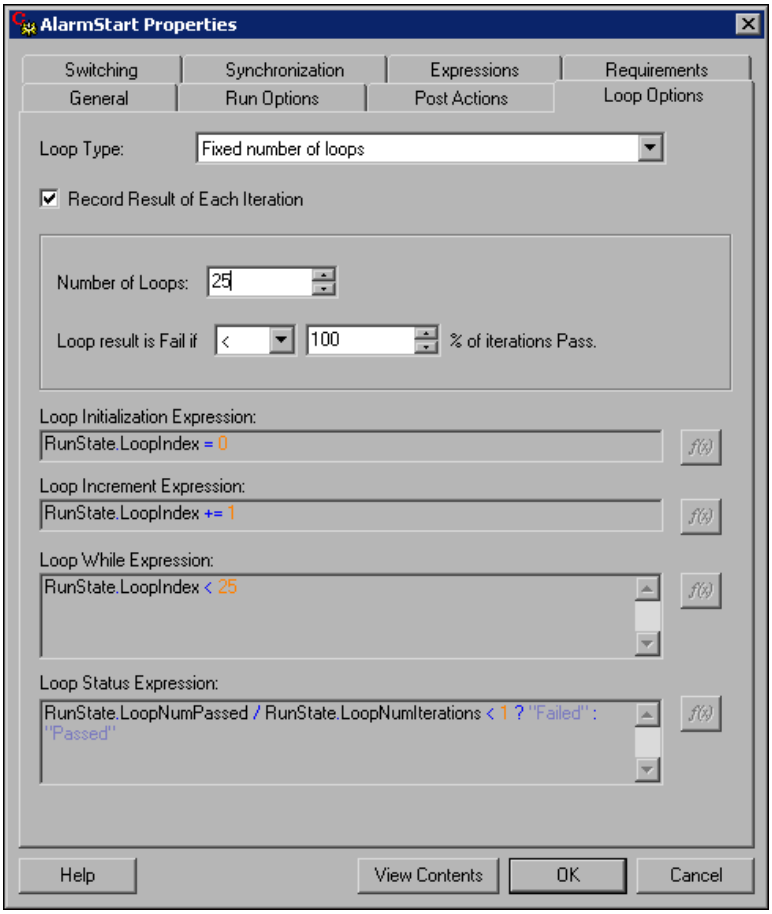


Figure 3-9. Loop Options Tab Settings

19. Click **OK** to close the Step Properties dialog box. The Sequence File window should now be the active window.
20. Create another Action step by right-clicking the `AlarmStart` step and selecting **Insert Step»Action** from the context menu. Name the new step `AlarmStop`.

21. Right-click the `AlarmStop` step and select **Specify Module** from the context menu. Repeat steps 6 through 10, but select the function shown in Figure 3-10. This step stops the beep that the AlarmStart step sets off.

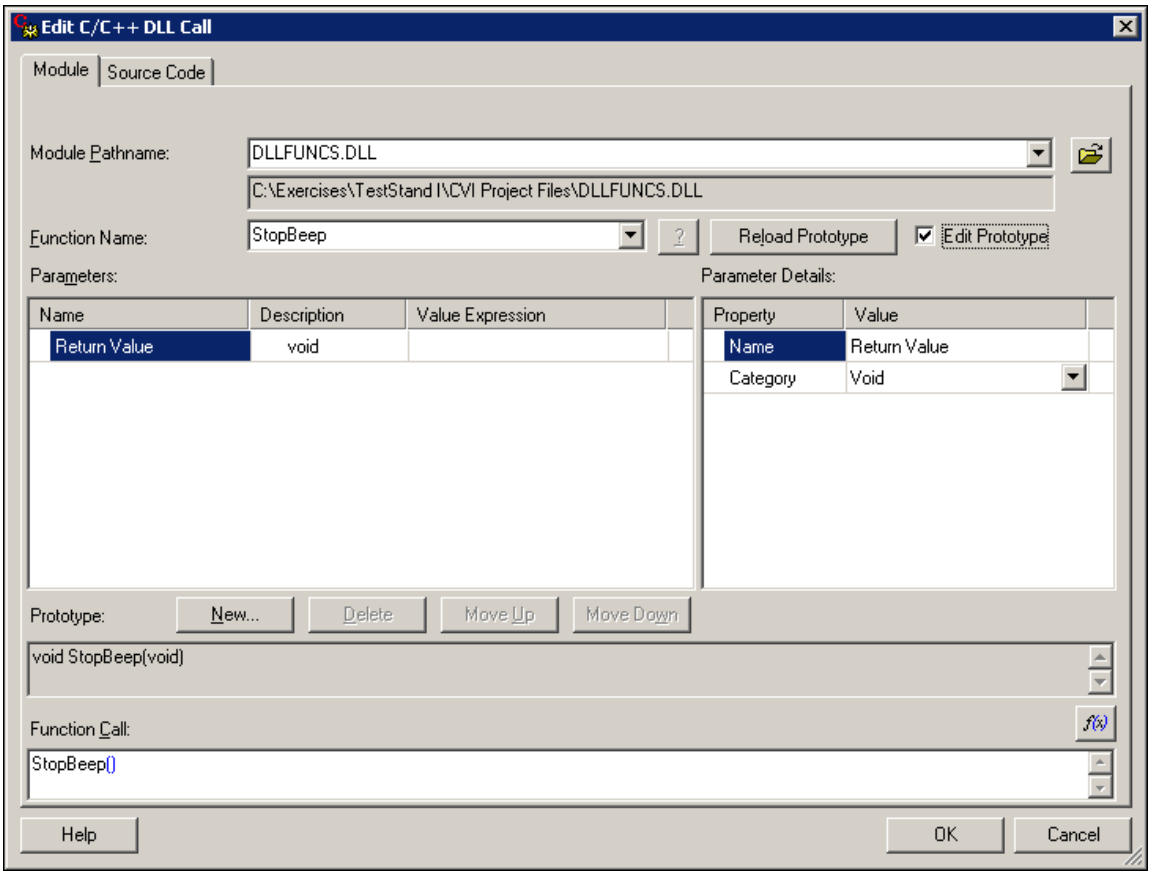


Figure 3-10. Edit C/C++ DLL Call Dialog Box

22. Save the sequence file.
23. Select **Execute»Test UUTs** and execute the sequence file. You should hear the PC speaker beep if you set the ROM test to fail in the Test Simulator dialog box.

Note Some PCs might not have a speaker. If the sound is not enabled on the machine, and you are taking this course at a National Instruments training facility, ask the instructor for assistance.

Note If execution tracing is disabled or the tracing speed is too high, the beeps may be played too fast to hear. You can configure execution tracing under the **Execution** tab of the **Station Options** configuration dialog box.

End of Exercise 3-1

Exercise 3-2 Configuring Loop Options

Objective: To build a sequence that loops on a step depending on the step results.

In TestStand, there are many different ways to choose to loop on a step, and there are many different results you can set at the end of the loop. This exercise examines some loop types and results using the Loop Options sequence execution.

1. In the sequence editor, select **File»New Sequence File**. Select **File» Save As** to save the empty sequence file as `Ex 3-2 Loop Step On Fail.seq` in the `C:\Exercises\TestStand I` directory.
2. Select the **LabVIEW Adapter** from the **Adapter** ring control, as shown in Figure 3-11.

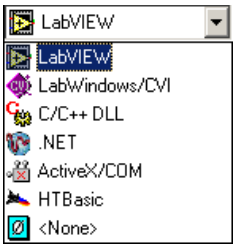


Figure 3-11. Adapter Ring Control

3. Right-click the Sequence File window and select **Insert Step»Tests» Numeric Limit Test** from the context menu.
4. Name the new step `RandomNumber`.

   This step tests random numbers between 0 and 1 returned from the code module. The test limit checks the results, and if the value is less than 0.5, indicating a failure, the step repeats.
5. Right-click the `RandomNumber` step and select **Specify Module** from the context menu.
6. In the Edit LabVIEW VI Call dialog box, shown in Figure 3-13, click the **File Browse** button, shown at left. Select the Random Number VI located in the `C:\Exercises\TestStand I\VIs` directory.

   If TestStand prompts you to resolve the path to the Random Number VI, select the **Use a relative path for the file you selected** option.
7. If you see a VI Has Changed dialog box, this indicates that your version of LabVIEW is higher than the version used to save the VI. Choose **Yes** to open LabVIEW and then select **File»Save** and **File»Exit**. If you do not see this dialog, proceed to step 8.

Note If you see the VI Has Changed dialog box, you will most likely see the same dialog each time you call a VI from TestStand. Each time this occurs, save the VI in the current version of LabVIEW by repeating the instructions in step 7.

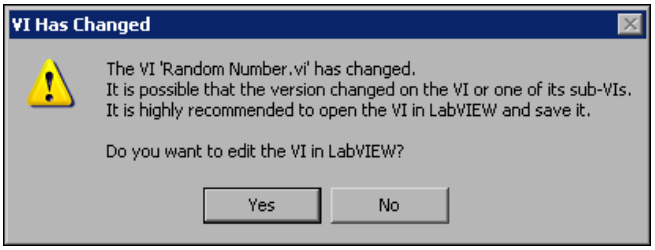


Figure 3-12. VI Has Changed dialog box

8. In the **Value** field of the **Numeric Measurement** item, enter `Step.Result.Numeric`. When you enter a period (`.`), TestStand automatically creates a pull-down menu of all the properties. Figure 3-13 shows the completed Edit LabVIEW VI Call dialog box.

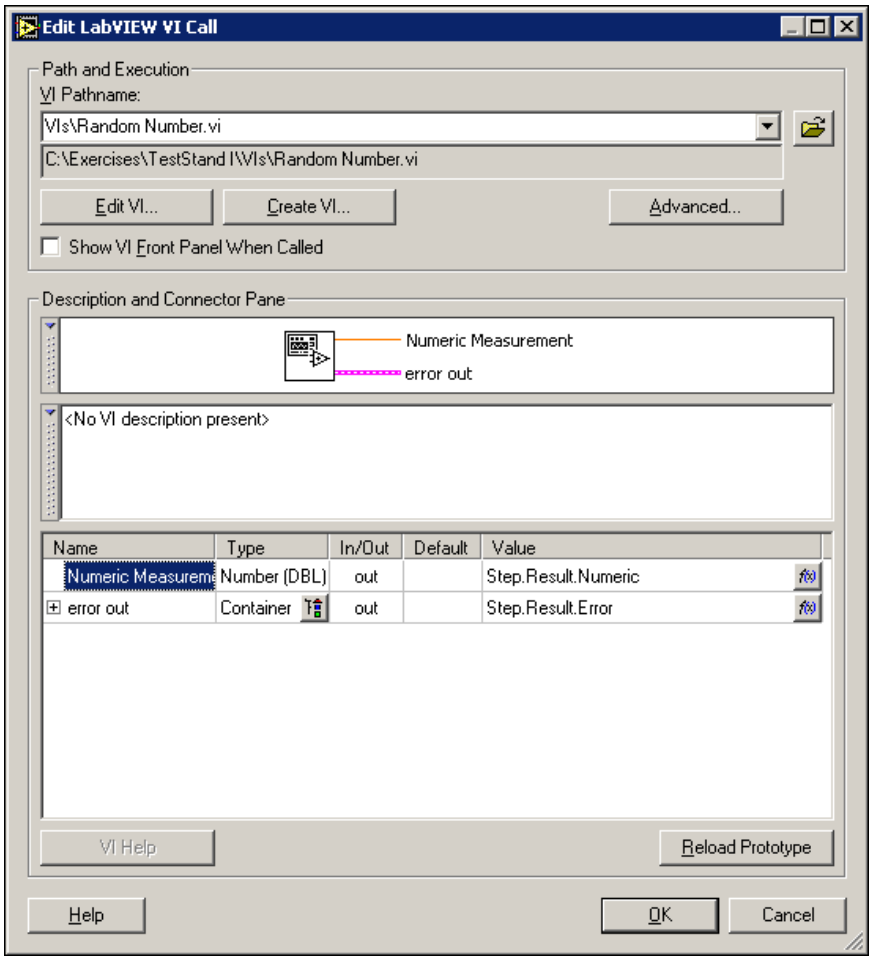


Figure 3-13. Edit LabVIEW VI Call Dialog Box

9. Click **OK** to close the Edit LabVIEW VI Call dialog box.
10. Right-click the `RandomNumber` step and select **Edit Limits** from the context menu. Set the limits in the Edit Numeric Limit Test dialog box as shown in Figure 3-14.

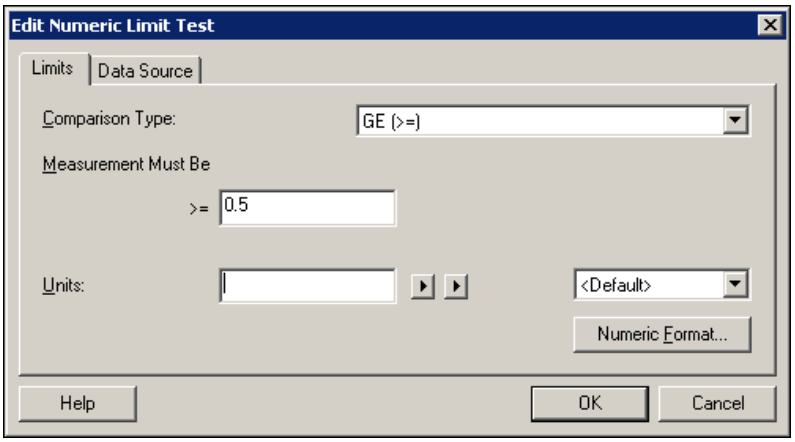


Figure 3-14. Edit Numeric Limit Test Dialog Box

Setting the limits in this way indicates `FAILED` if the returned measurement, random number, is less than 0.5 and `PASSED` if it is greater than or equal to 0.5. Click **OK** to close the Edit Numeric Limit Test dialog box.

11. Right-click the `RandomNumber` step and select **Properties** from the context menu to launch the Step Properties dialog box.
12. Click the **Loop Options** tab and set the loop options so that the step loops five times or until the test passes. Figure 3-15 shows the completed Loop Options tab.

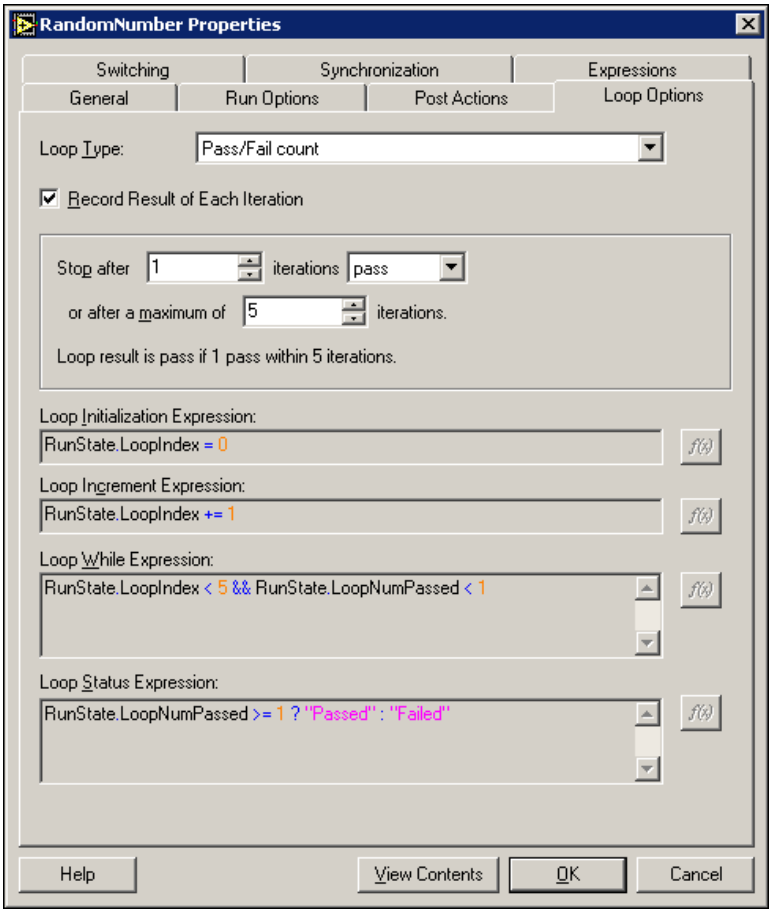


Figure 3-15. Loop Options Tab Settings

13. Click **OK** to close the Step Properties dialog box.
14. The sequence is now set up to run as desired. Run the sequence several times and examine the report each time. The report should look similar to the report shown in Figure 3-16.

Note The report is arranged to display a summary about the execution before displaying information about each loop iteration. If the RandomNumber step had been the third step, it would be the third item listed in the report and would be followed by the results of the fourth step.

In the test report shown in Figure 3-16, the RandomNumber step eventually passed. Notice that the first RandomNumber in the report contains more information than the others, including the number of passes, failures, and the total number of iterations of the step. Because the RandomNumber step had looping options set, the next item in the report is the result of each iteration of the loop.

RandomNumber	
Status:	Passed
Measurement:	0.7369202934001
Limits:	
Low:	0.5
Comparison Type:	GE (>=)
Module Time:	0.2378911
Number of Loops:	2
Number of Passes:	1
Number of Failures:	1
Final Loop Index:	2
RandomNumber (Loop Index: 0)	
Status:	Failed
Measurement:	0.4816512844077
Limits:	
Low:	0.5
Comparison Type:	GE (>=)
Module Time:	0.2213024
RandomNumber (Loop Index: 1)	
Status:	Passed
Measurement:	0.7369202934001
Limits:	
Low:	0.5
Comparison Type:	GE (>=)
Module Time:	0.0165887

Figure 3-16. Random Number Test Report

15. Save and close the finished sequence file before you start the next exercise.

End of Exercise 3-2

Exercise 3-3 Creating a Sequence

Objective: To create a sequence using different step types.

This exercise builds on skills you have learned in previous lessons. In the first part, you create a sequence to be used as a subsequence. The second part shows how to use steps created using different adapter types together in the same sequence.

Part A: Creating the CPU Subsequence

1. In the sequence editor, select **File»Open** and open the `Ex 3-3 Creating a Sequence File.seq` file located in the `C:\Exercises\TestStand I` directory.
2. Right-click the `Powerup Test` step and select **Insert Step»Sequence Call** from the context menu. Figure 3-17 shows the context menus associated with inserting a Sequence Call step.

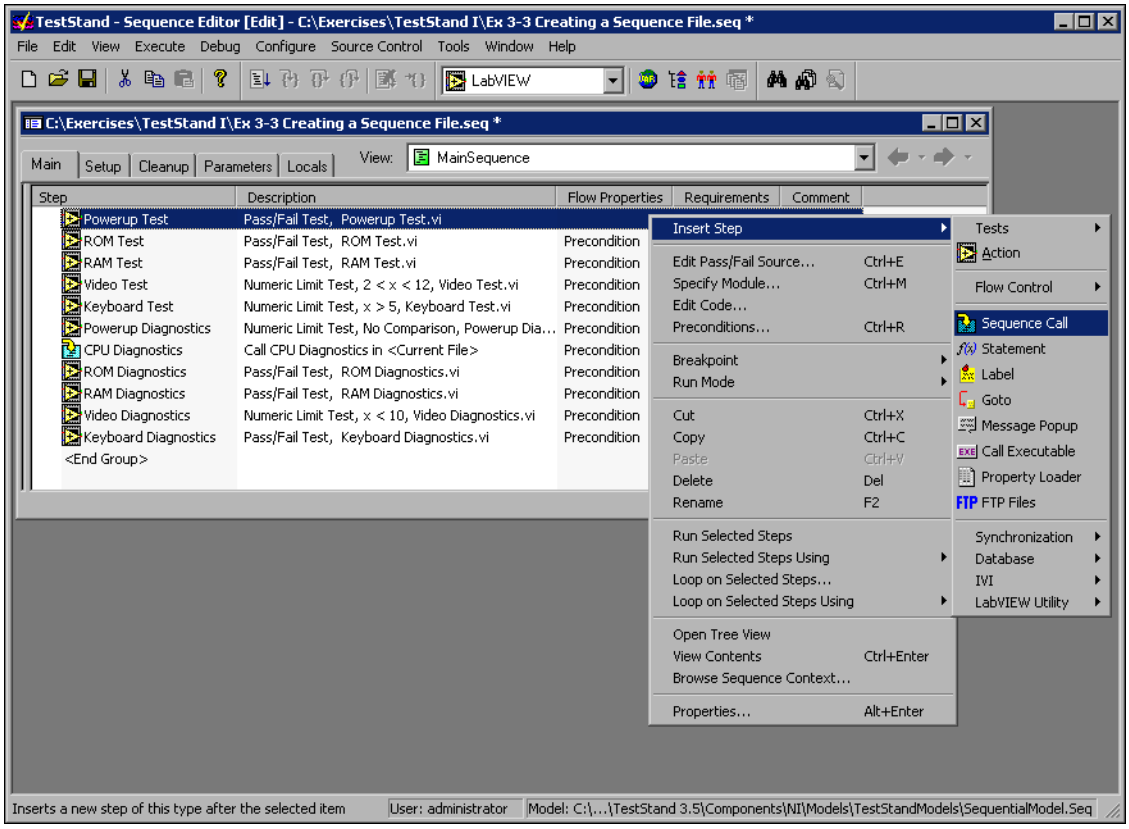


Figure 3-17. Inserting a Sequence Call Step

3. Rename the step `CPU Test`.

   This step calls a specified sequence file as a subsequence during sequence execution. For the subsequence call to work, there must be an existing sequence for this step to call. Therefore, the next step is to create the sequence, `CPU Test`, to call as the subsequence.

Note The step and sequence to be called do not need to have the same name. However, for this exercise it is easier to remember the sequence name if the step name is the same.

4. Select **All Sequences** from the **View** ring control as shown in Figure 3-18.

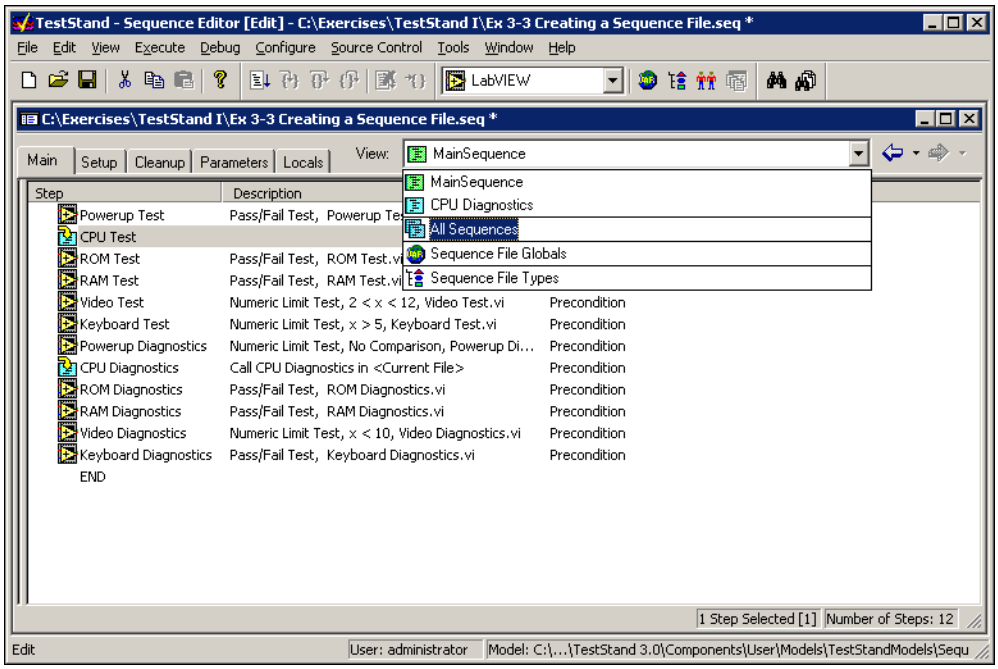


Figure 3-18. View Ring Control

5. Notice that there is not a sequence named CPU Test in this sequence file. Create this sequence in the sequence file by right-clicking the window and selecting **Insert Sequence** from the context menu, as shown in Figure 3-8.

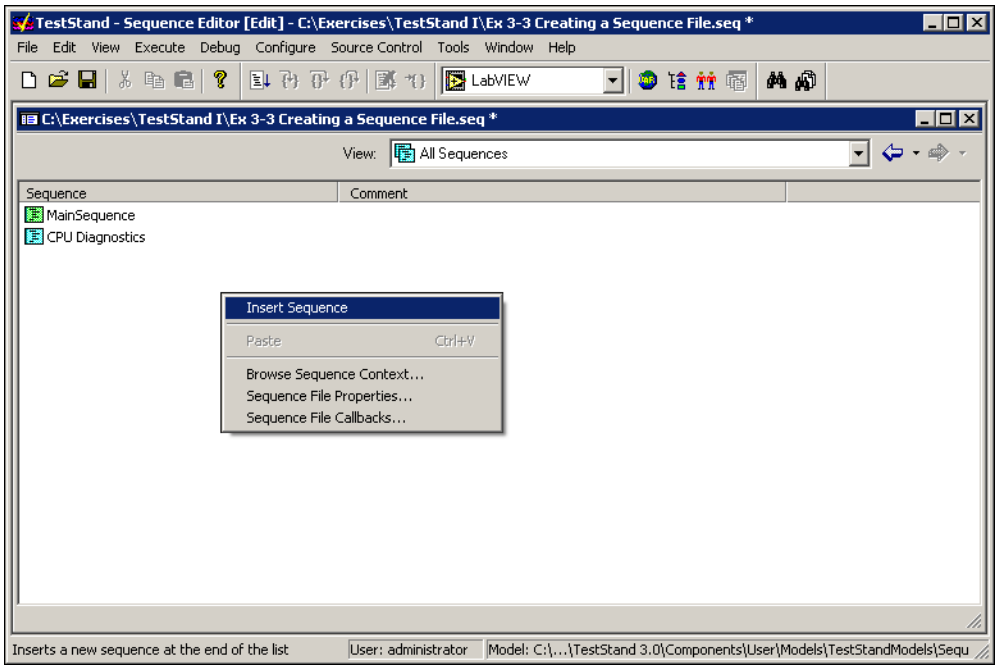


Figure 3-19. Inserting a New Sequence

6. Rename the new sequence `CPU Test`.
7. Select **CPU Test** from the **View** ring control to open the `CPU Test` sequence. This sequence does not contain any steps. This is the sequence you will create.
8. If the LabVIEW Adapter is not selected, select it from the **Adapter** ring control.
9. Click the **Setup** tab to view the steps in the Setup step group. The setup for this subsequence requires one function.
10. Right-click in the Setup step group and select **Insert Step»Action** from the context menu, as shown in Figure 3-20.

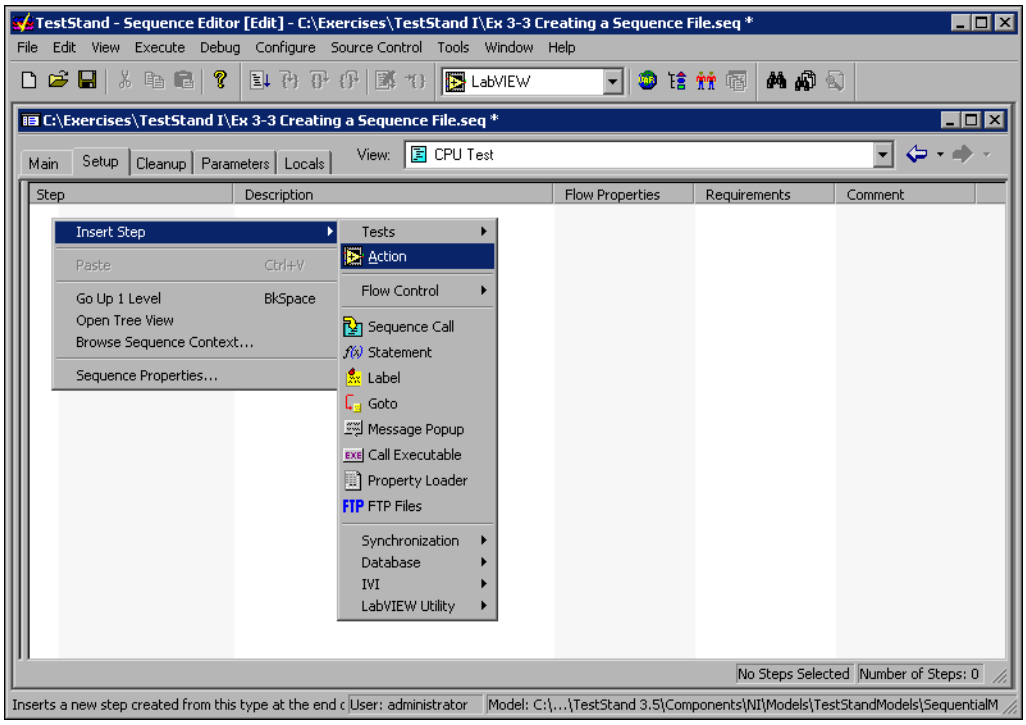


Figure 3-20. Inserting an Action Step

11. Rename the Action step `Pick Test To Fail`.
12. Right-click the step and select **Specify Module** from the context menu to launch the Edit LabVIEW VI Call dialog box.
13. Click **File Browse**, shown at left, and select the CPU PreTest VI located in the `C:\Exercises\TestStand I\VIs` directory.
14. Click **OK** to select this VI as the code module.
15. If a dialog box prompts you to select an option for the file path, select the **Use a relative path for the file you selected** option. This option makes the source code relative to the sequence file. Click **OK** to return to the Edit LabVIEW VI Call dialog box.
16. If you see a VI Has Changed dialog box, this indicates that your version of LabVIEW is higher than the version used to save the VI. Choose **Yes** to open LabVIEW and then select **File»Save** and **File»Exit**. If you are prompted to save additional files, choose **Save SubVIs**. If you do not see the VI Has Changed dialog, proceed to step 17.

Note If you see the VI Has Changed dialog box, you will most likely see the same dialog each time you call a VI from TestStand. Each time this occurs, save the VI in the current version of LabVIEW by repeating the instructions in step 16.

Figure 3-21 shows the completed Edit LabVIEW VI Call dialog box.

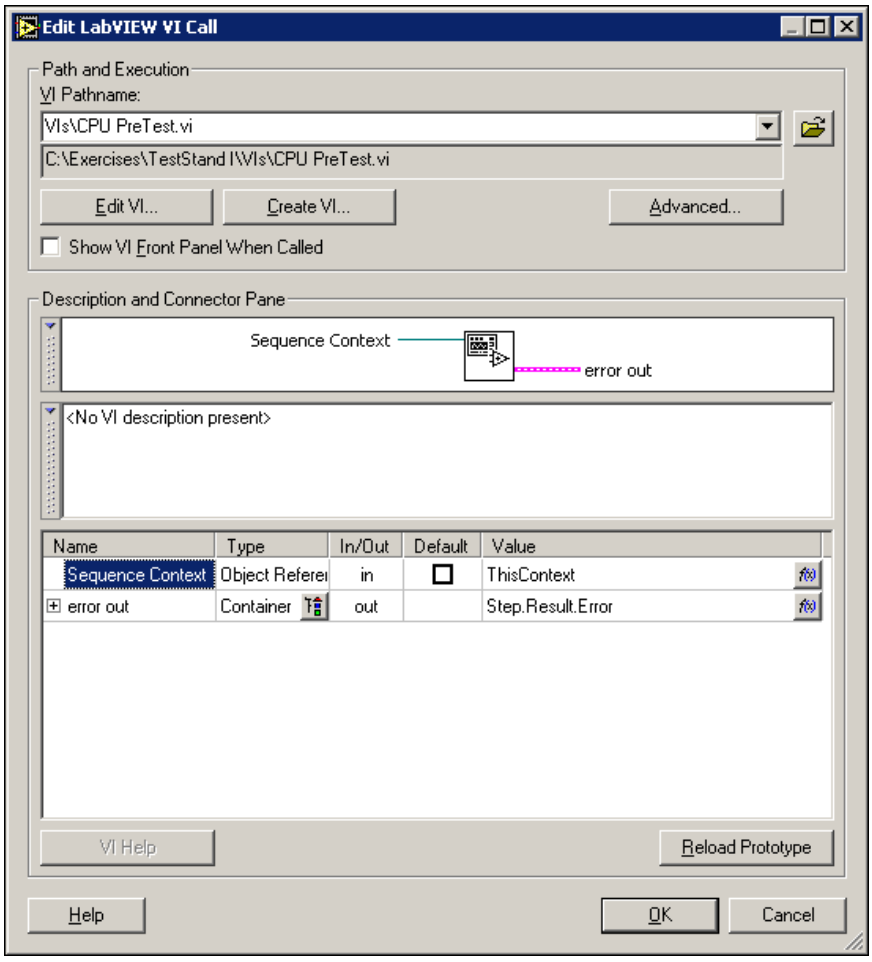


Figure 3-21. Edit LabVIEW VI Call Dialog Box

17. Click **OK** to return to the Sequence File window.
18. Click the **Main** tab to display the Main step group. Right-click in the Main step group and select **InsertStep»Tests»Pass/Fail Test** from the context menu, as shown in Figure 3-22.

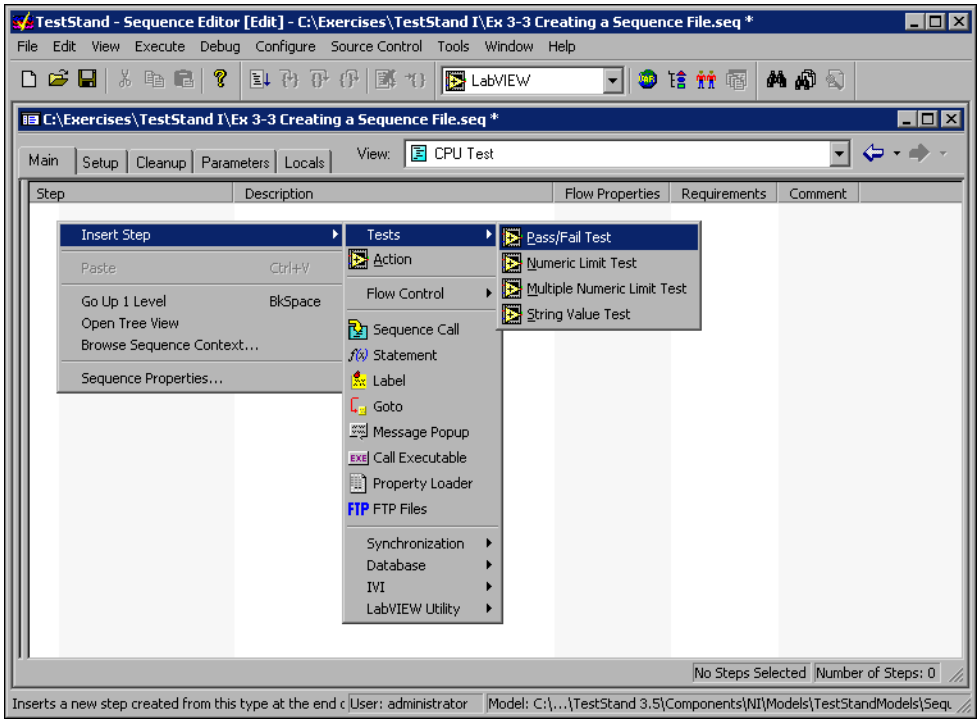


Figure 3-22. Inserting a Pass/Fail Test

19. Rename the step `Register Test`.
20. Repeat step 18 three times to create the `Instruction Set Test`, `Cache Test`, and `FPU Test` steps, as shown in Figure 3-23.

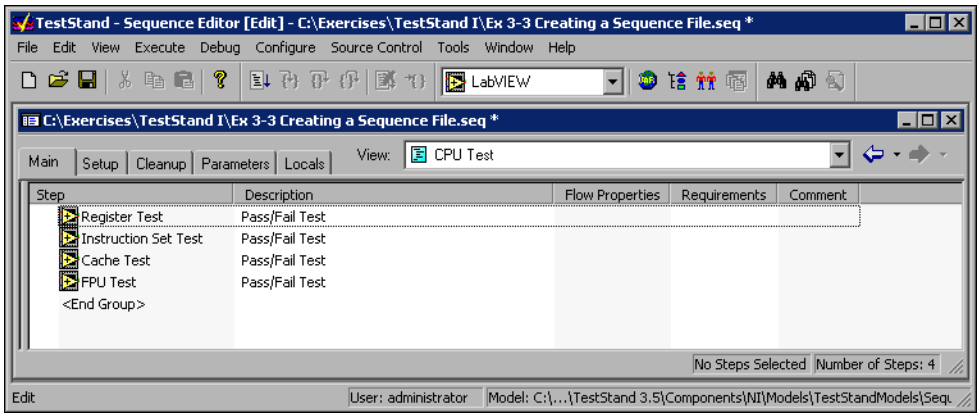


Figure 3-23. Creating Test Steps

21. Right-click the `Register Test` step and select **Specify Module** from the context menu to launch the Edit LabVIEW VI Call dialog box.
22. Click **File Browse** to select the Register Test VI code module, located in the `C:\Exercises\TestStand I\VIs` directory.

    If TestStand prompts you to resolve the path to the Register Test VI, select the **Use a relative path for the file you selected** option.

23. Click **OK** to return to the Edit LabVIEW VI Call dialog box.
    a. Enter `Step.Result.PassFail` in the PASS/FAIL Flag **Value** field.
    b. Enter `Step.Result.ReportText` in the Report Text **Value** field.

    Figure 3-24 shows the completed Edit LabVIEW VI Call dialog box.

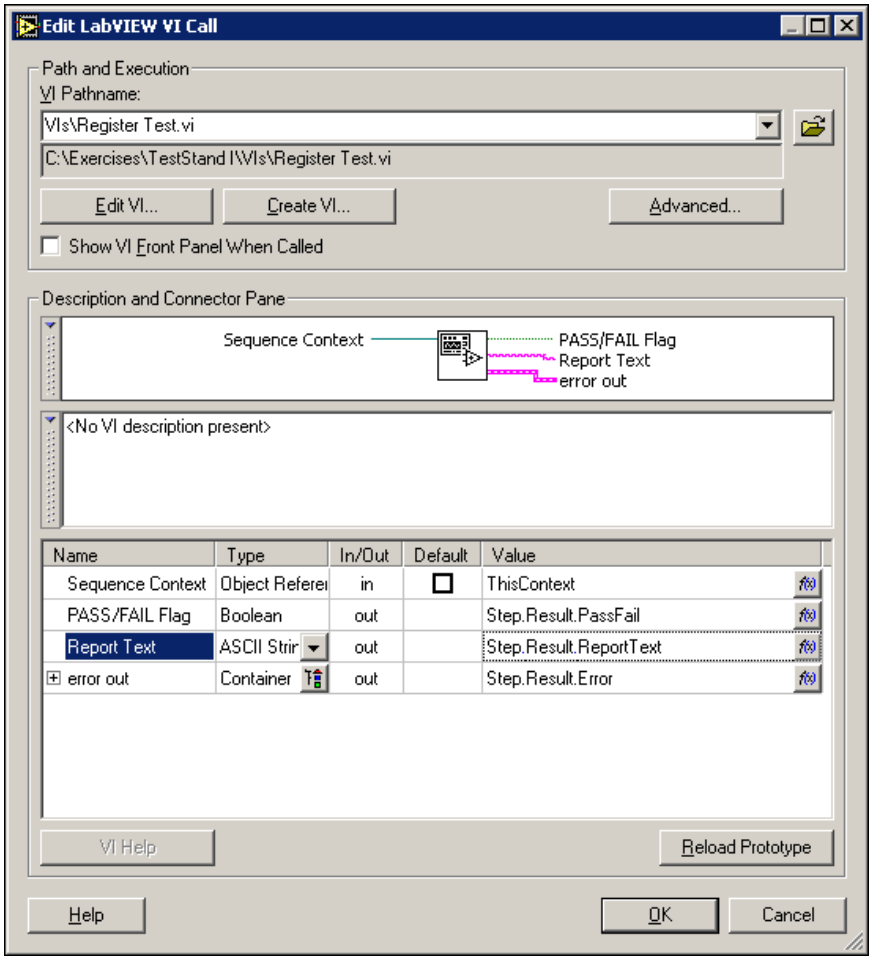


Figure 3-24. Edit LabVIEW VI Call Dialog Box

    c. Click **OK** to close the Edit LabVIEW VI Call dialog box.

24. Repeat steps 21 through 23 for the `Instruction Set Test` step, selecting the Instruction Set Test VI in step 22.
25. Repeat steps 21 through 23 for the `Cache Test` step, selecting the Cache Test VI in step 22.
26. Repeat steps 21 through 23 for the `FPU Test` step, selecting the FPU Test VI in step 22. Figure 3-25 shows the completed window.

The name of the code module called at each step appears in the **Description** column.

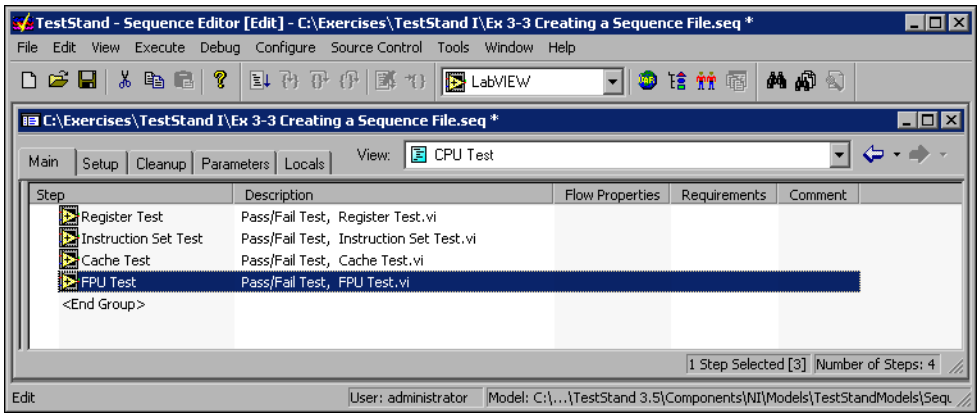


Figure 3-25. Code Modules in Description Column

27. Click the **Parameters** tab and right-click in the right window pane. Select **Insert Parameter»Boolean** from the context menu to create a Boolean parameter that passes into this sequence when it is called as a subsequence. Figure 3-26 shows the context menus associated with inserting a parameter. Refer to Lesson 4, *TestStand Parameters, Variables, and Expressions*, for more information about parameters.

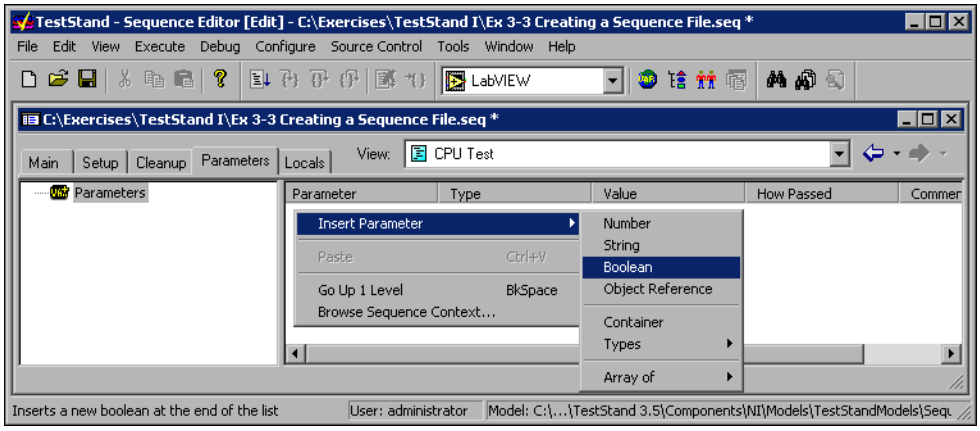


Figure 3-26. Inserting a Boolean Parameter

28. Rename the new parameter `CPUFail`.
29. Save the sequence.
30. Select the `MainSequence` from the **View** ring control.
31. Click the **Main** tab to view the Main step group of the `MainSequence`.
32. Right-click the `CPU Test` step and select **Specify Module** from the context menu to select which sequence to call at this step.

33. In the Edit Sequence Call dialog box that launches, enable the **Use Current File** option and select the **CPU Test** sequence from the **Sequence** ring control. Make sure the **Use Prototype of Selected Sequence** option is enabled, as shown in Figure 3-27.

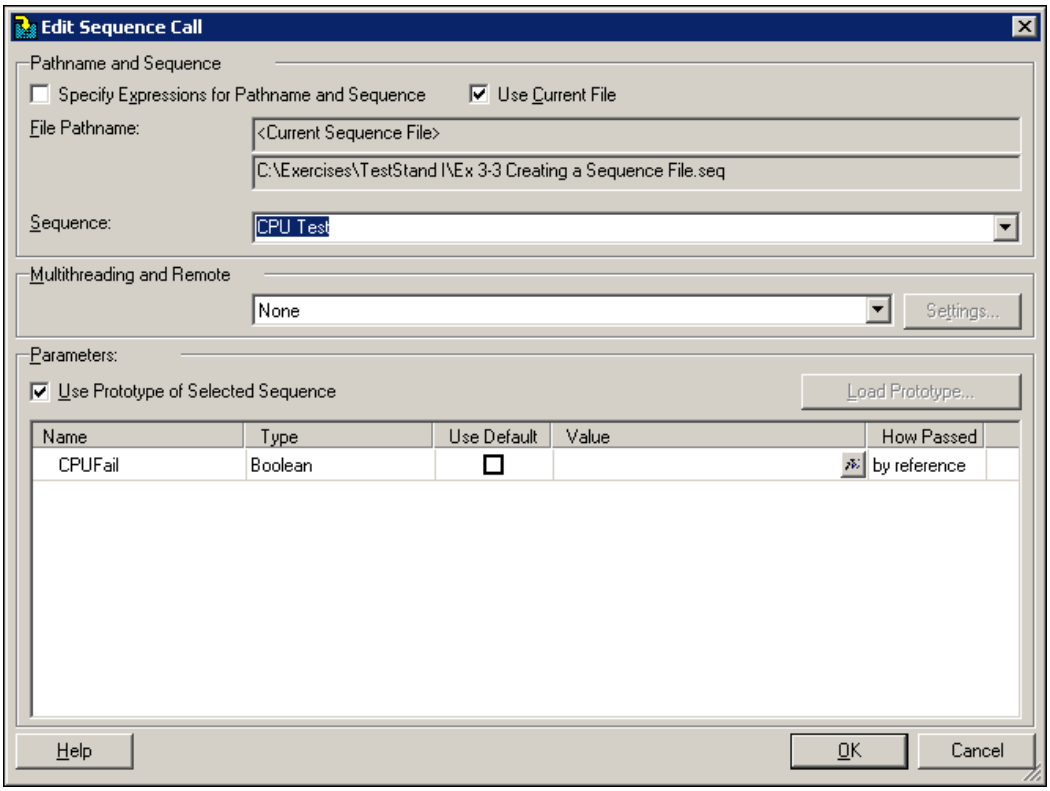


Figure 3-27. Selecting a Sequence to Call

Additional Information

You have selected the CPU Test sequence you created during this exercise as the sequence to call at this step. Remember that this sequence requires the CPUFail parameter to be passed to it. Notice that when you select the CPU Test sequence, TestStand automatically adds the CPUFail parameter to the list of parameters to be passed to the sequence because you enabled the Use Prototype of Selected Sequence option.

34. Specify the value for this parameter. Click the **Value** column for the `CPUFail` parameter, then click the **Expression Browse** button to launch the Expression Browser dialog box.

35. In the Expression Browser dialog box, click the **+** sign next to `Locals`, then click the `CPUFail` variable. Click **Insert** to select that expression as the value to be passed. Figure 3-28 shows the completed Expression Browser dialog box.

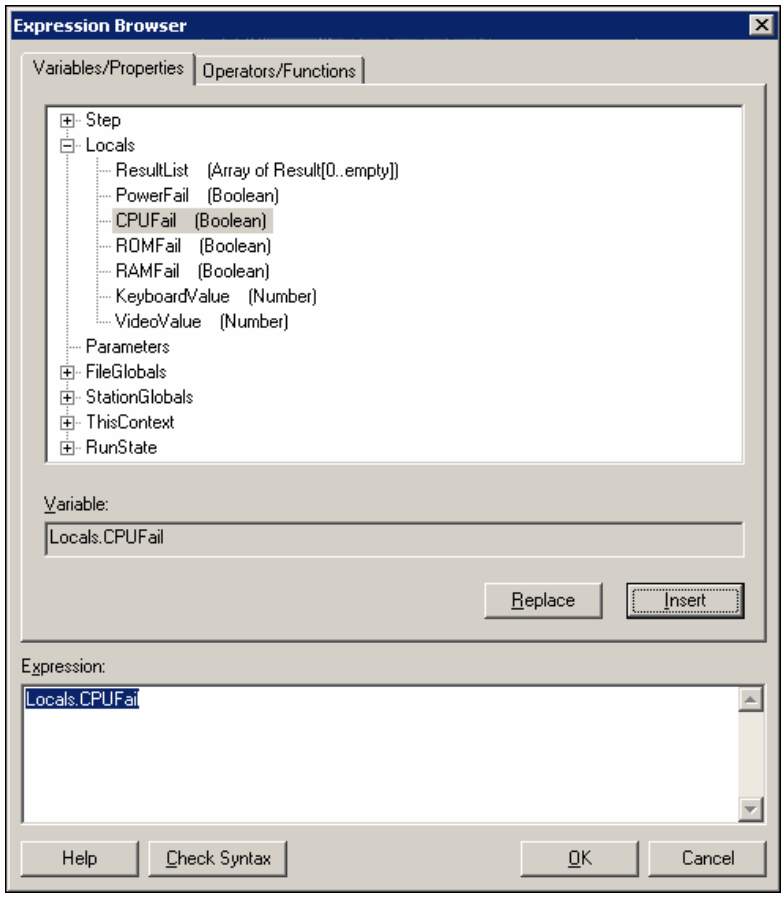


Figure 3-28. Expression Browser Dialog Box

36. Click **OK** to return to the Edit Sequence Call dialog box, which should appear as shown in Figure 3-29.

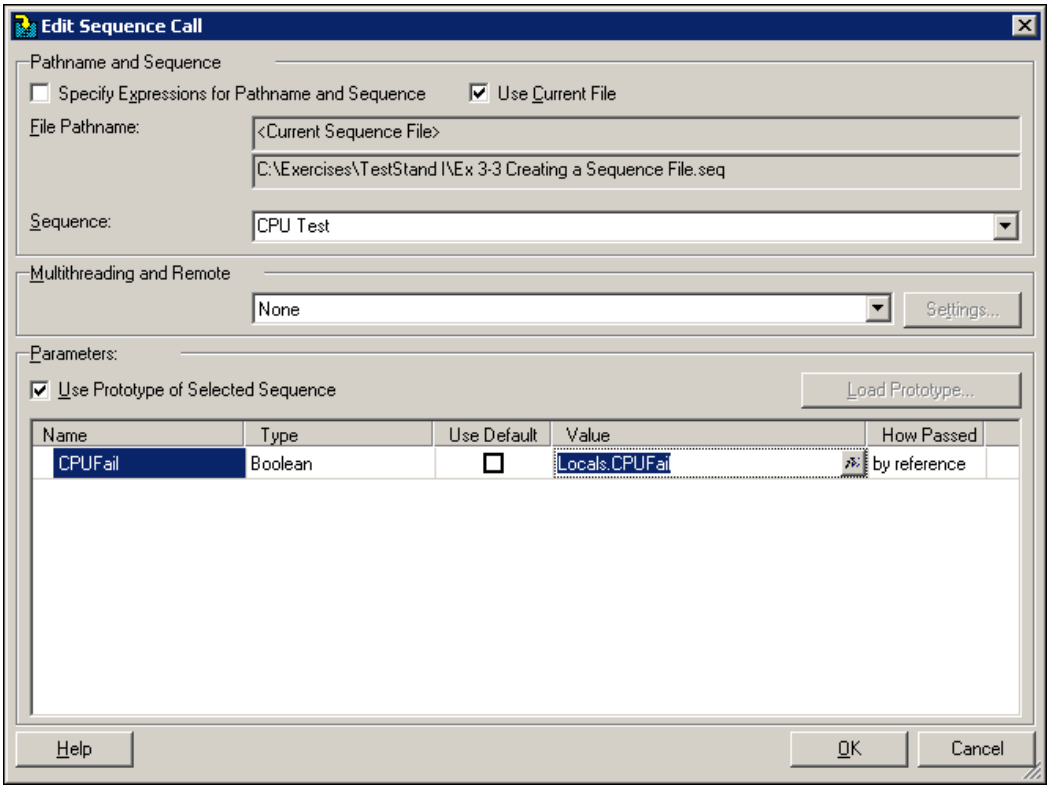


Figure 3-29. Edit Sequence Call Dialog Box

37. Click **OK** to return to the Sequence File window.
38. Save the changes to the sequence file.
39. Select **Execute»Test UUTs** or click **Run** on the toolbar to run the sequence.

    The report now contains a section showing that the `CPU Test` sequence ran. Another section lists the results of each of the four tests you created in this exercise.
40. Close the report, but leave the sequence file open for the next part of this exercise.

End of Part A

Part B: Using Different Adapter Types in the Same Sequence

So far, each sequence you have constructed has used the same module adapter type. However, you can use different module adapters in the same sequence. This part of the exercise shows you how to modify the `CPU Test` sequence so that it uses different module adapters for some of the steps. Then, you will run the sequence again to show that the only change in the sequence behavior is where the sequence searches for the code resources.

1. Select the `CPU Test` sequence from the **View** ring control.
2. Right-click the `Instruction Set Test` step and select **Properties** from the context menu to display the Step Properties dialog box.
3. Select the **LabWindows/CVI Adapter** from the **Adapter** ring control to change the module adapter, as shown in Figure 3-30.

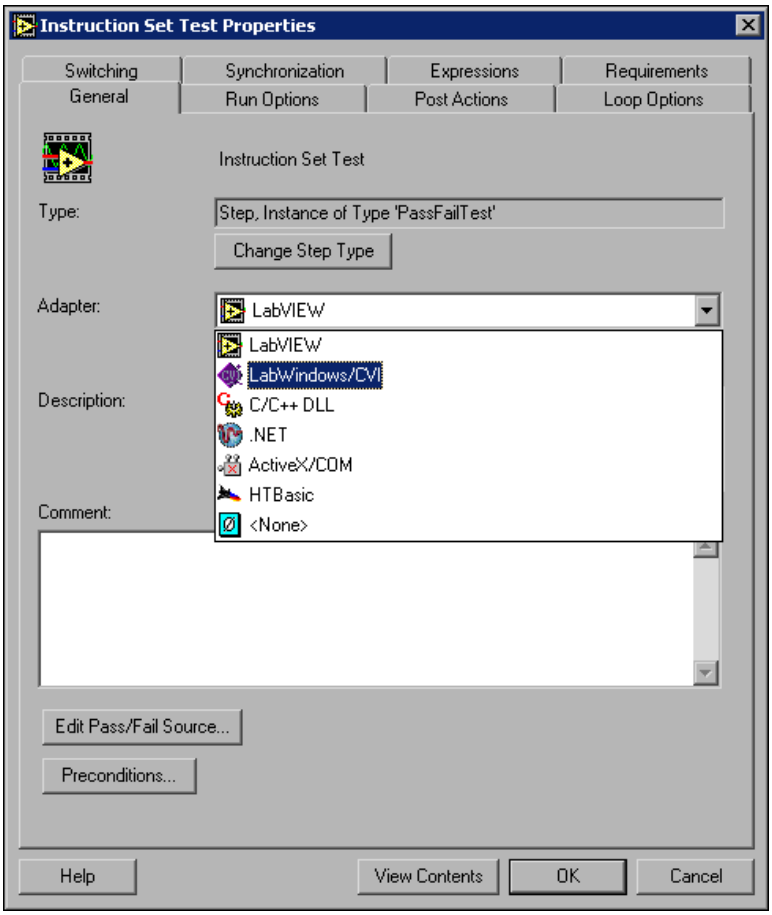


Figure 3-30. Adapter Ring Control

Note Instead of deleting steps and adding new steps that use a different module adapter or step type, you can use the Step Properties dialog box to change the adapter or step type of an existing step. When you use the Step Properties dialog box to change the module

adapter or step type, other settings, such as loop options, preconditions, and post actions, do not change.

4. Click **OK** to close the Step Properties dialog box and return to the Sequence File window.
5. Right-click the `Instruction Set Test` step and select **Specify Module** from the context menu.
6. Click **File Browse** and select the `Ex3-3.dll` code module located in the `C:\Exercises\TestStand I\CVI Project Files` directory.

   If TestStand prompts you to resolve the path to the file, select the **Use a relative path for the file you selected** option.
7. Select the **InstrSetTest** function from the **Function Name** ring control. The parameters load automatically because the function prototypes are exported through a type library.
8. Enter the function parameters shown in the following table.

Parameter	Value Expression
seqContextCVI	`ThisContext`
result	`Step.Result.PassFail`
errorOccurred	`Step.Result.Error.Occurred`
errorCode	`Step.Result.Error.Code`
errorMsg	`Step.Result.Error.Msg`

Figure 3-31 shows the completed dialog box.

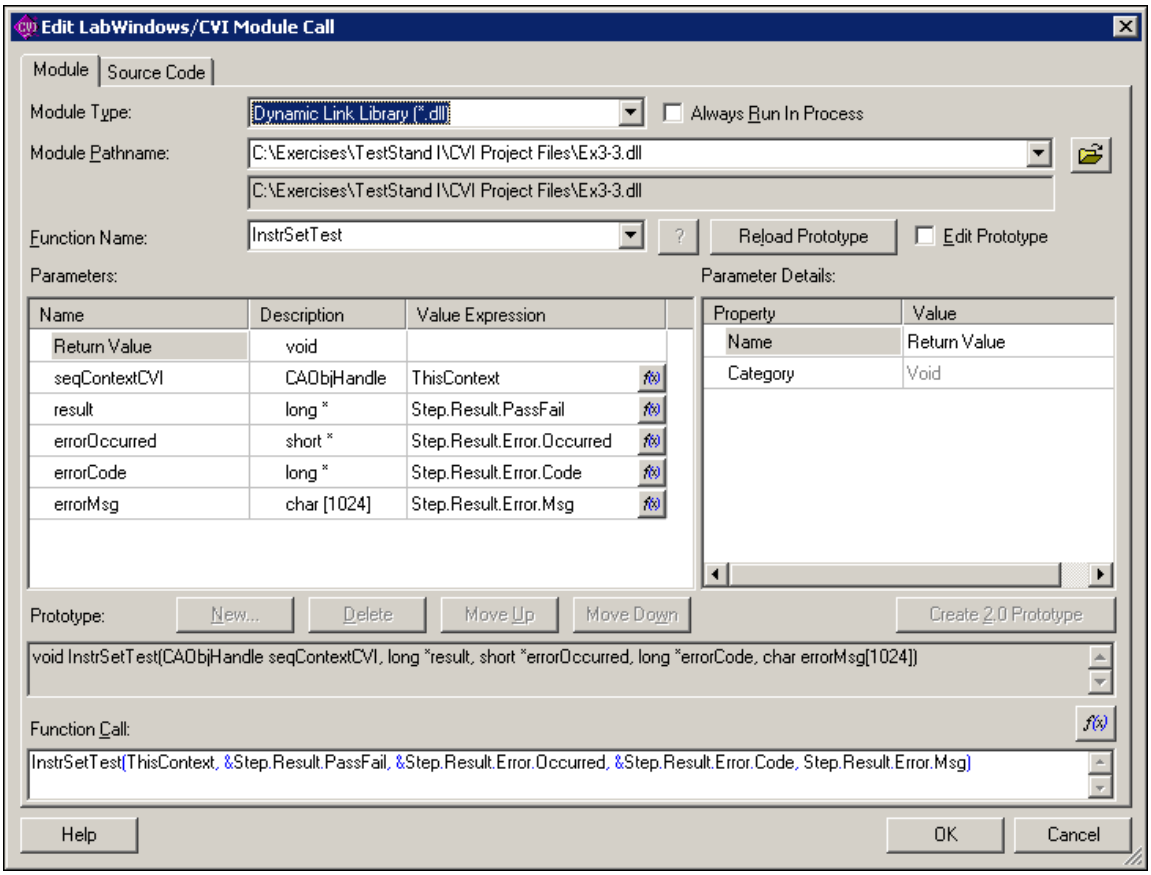


Figure 3-31. Edit LabWindows/CVI Module Call Dialog Box

9. Click **OK** to close the Edit LabWindows/CVI Module Call dialog box.
10. Right-click the `Cache Test` step and select **Properties** from the context menu to display the Step Properties dialog box.
11. Select the **C/C++ DLL Adapter** from the **Adapter** ring control as shown in Figure 3-32.

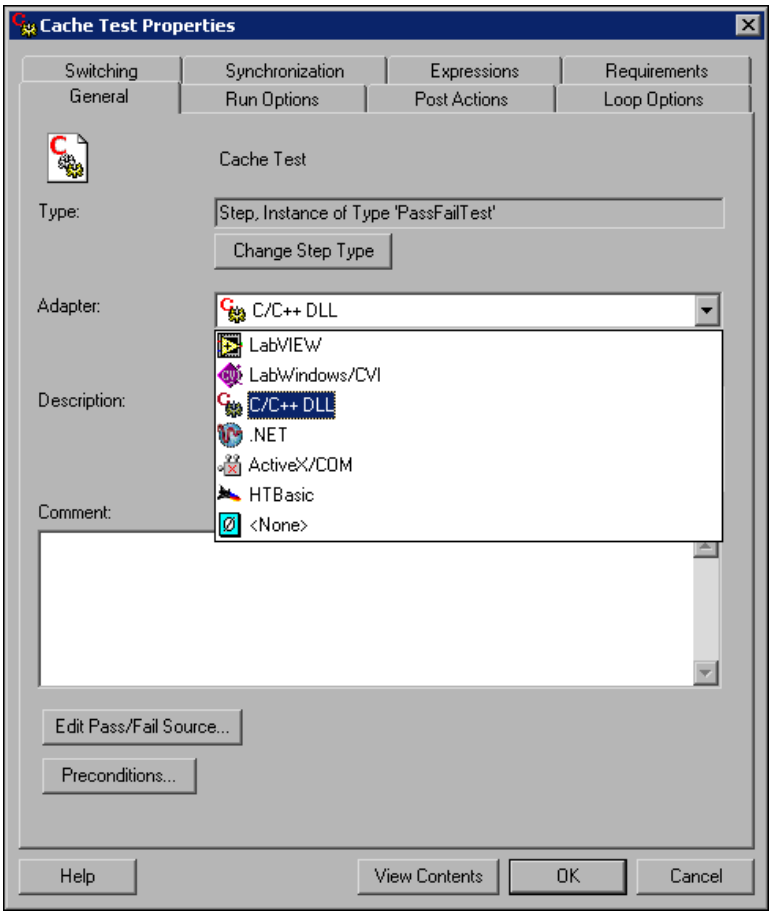


Figure 3-32. Adapter Ring Control

12. Click **OK** to return to the Sequence window.
13. Right-click the `Cache Test` and select **Specify Module** from the context menu to launch the Edit C/C++ DLL dialog box.
14. Click the **File Browse** button and select the `Ex3-3.dll` code module located in the `C:\Exercises\TestStand I\CVI Project Files` directory.
15. Select the **CacheTest** from the **Function Name** ring control.
16. Enter the function parameters shown in the following table.

Parameter	Value Expression
seqContextCVI	`ThisContext`
result	`Step.Result.PassFail`
errorOccurred	`Step.Result.Error.Occurred`
errorCode	`Step.Result.Error.Code`
errorMsg	`Step.Result.Error.Msg`

The Function Call section at the bottom of the window displays the actual function call. Figure 3-33 shows the completed Edit C/C++ DLL dialog box.

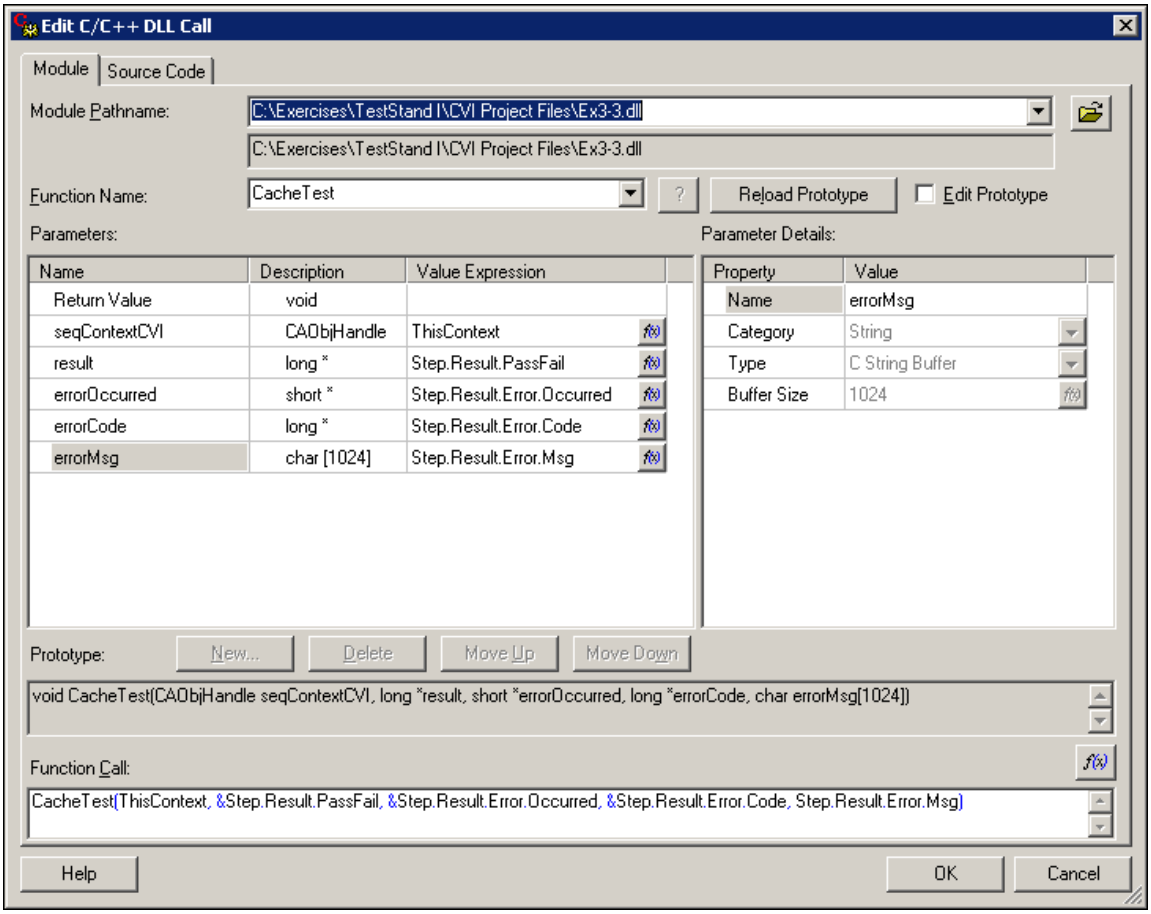


Figure 3-33. Edit C/C++ DLL Dialog Box

17. Click **OK** to exit the Edit C/C++ DLL dialog box.
18. Select `MainSequence` from the **View** ring control.
19. Save the sequence and select **Execute»Test UUTs** or click **Run** on the toolbar to run the sequence.

    The sequence file should execute the same as in it did Part A and an identical report should result. The only difference is that two of the steps in the `CPU Test` subsequence now call DLLs using different module adapter types instead of calling LabVIEW VIs using the LabVIEW Adapter.
20. Close the sequence file when you finish.

End of Part B

End of Exercise 3-3

Exercise 3-4 Running Simultaneous Sequences (Optional)

Objective: To show how TestStand can run sequences simultaneously.

TestStand can run multiple sequences simultaneously. Each sequence runs in a separate thread when it executes. This exercise demonstrates a simple example of simultaneous execution by showing how to run the same sequence multiple times in both the sequence editor and one of the operator interfaces. It is possible to run different sequences at the same time instead of running the same sequence many times simultaneously. Refer to the *TestStand II: Customization Course Manual* for more information about the additional multithreading features in TestStand.

Note For this exercise, make sure you select the **Execute»Tracing Enabled** option or select **Configure»Station Options** and adjust the **Speed Slider** control on the **Execution** tab. When you enable this option, the sequence execution includes a yellow pointer icon indicating the current sequence execution step. When this option is not enabled, there are no visible markers to indicate the current step is executing in any Execution window.

1. In the sequence editor, open the `runforever.seq` sequence file located in the `<TestStand>\Examples\Demo\Misc` directory.

   Figure 3-34 shows the example that ships with TestStand. The `runforever.seq` sequence file runs a series of steps then loops back to the beginning of the sequence and runs the steps again. This process continues until you terminate the sequence manually.

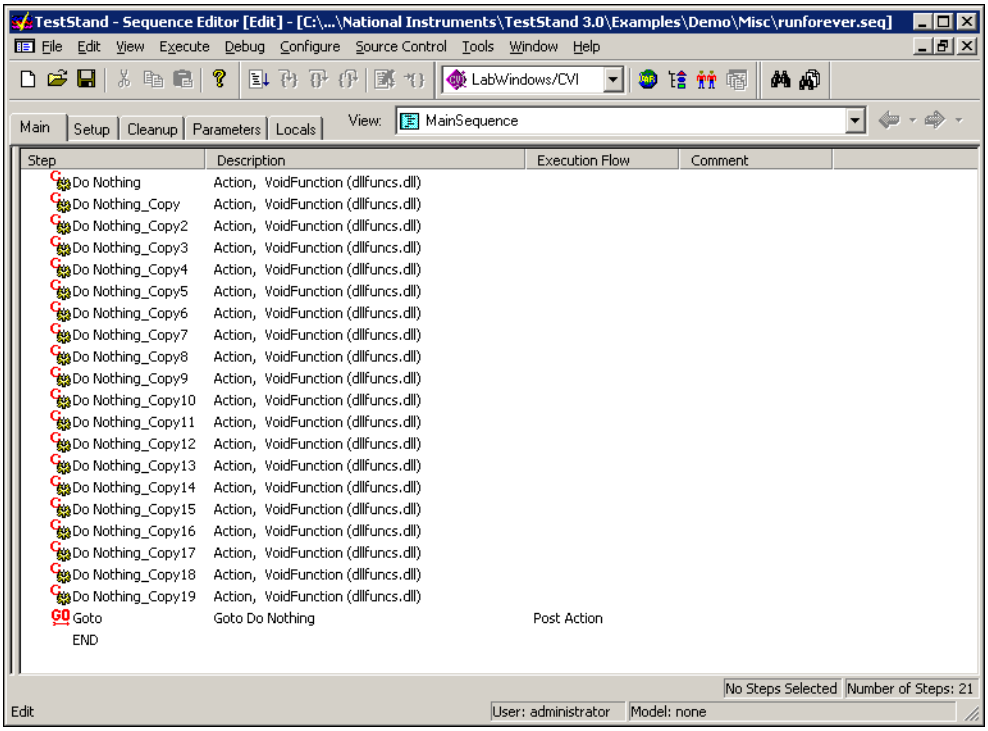


Figure 3-34. RunForever Sequence File

2. Move and resize the Sequence File window so you can view multiple windows in the sequence editor, as shown in Figure 3-35.

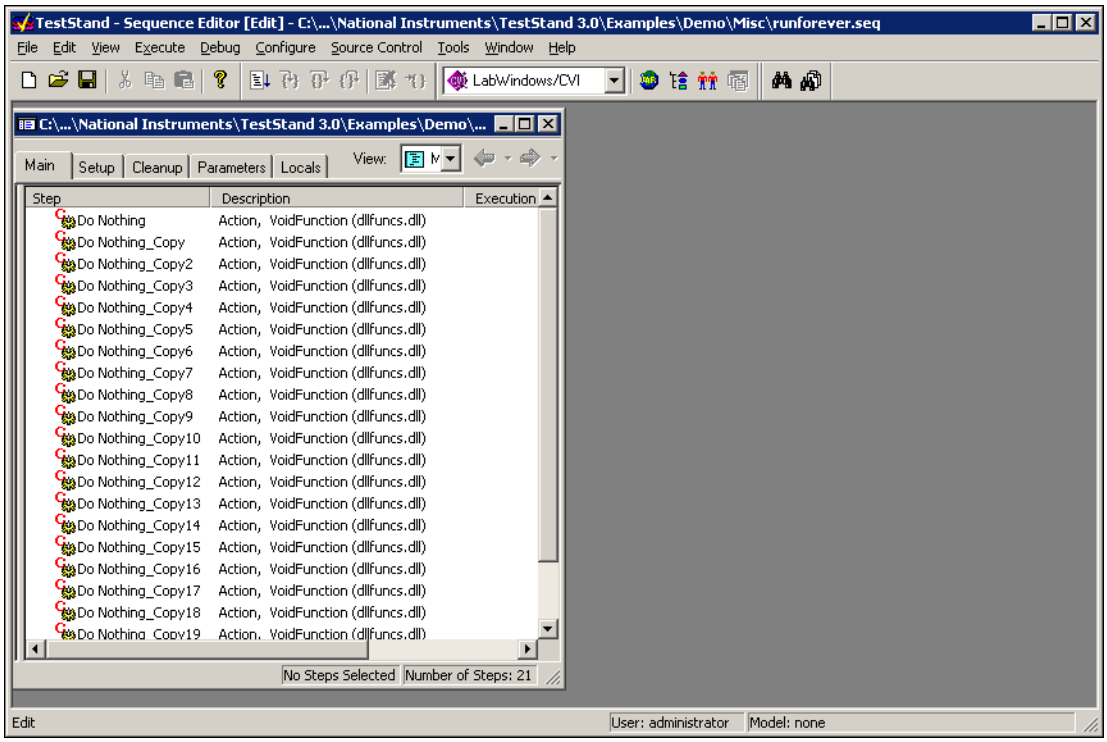


Figure 3-35. Modified Sequence Display Window

3. Click **Run** on the toolbar and notice that the sequence begins executing and continues running when it reaches the end of the sequence file. Figure 3-36 shows the sequence file during execution.

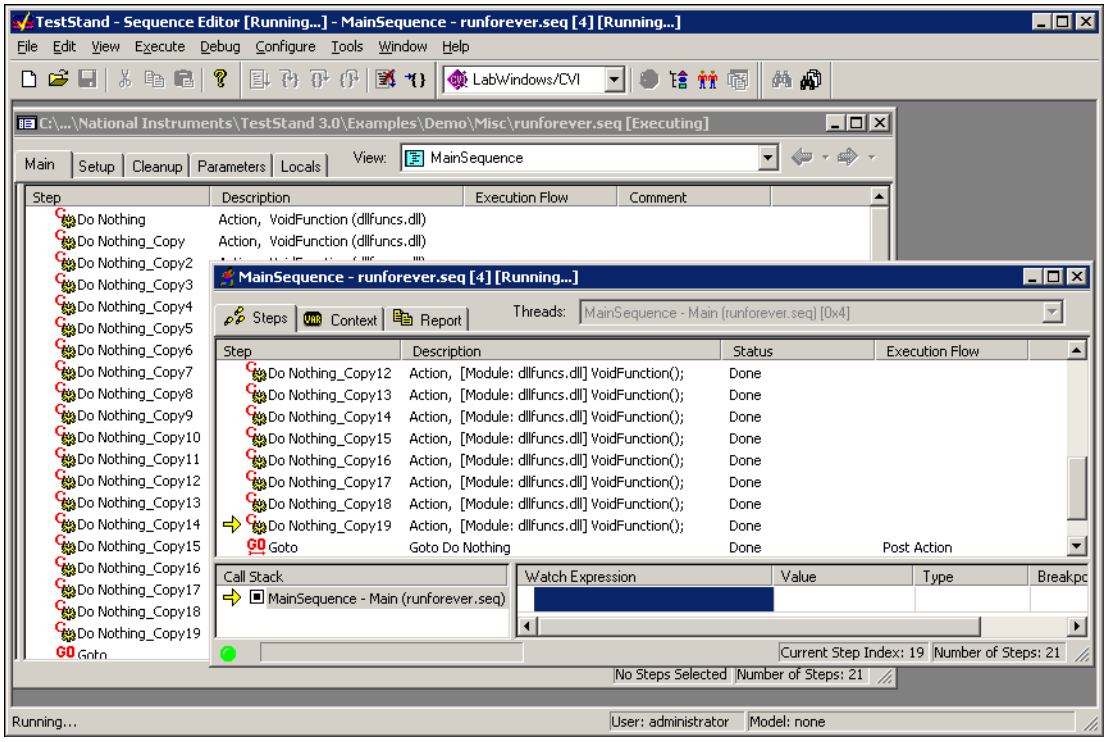


Figure 3-36. Executing the RunForever Sequence

4. Click the **runforever.seq [Executing]** window, shown in Figure 3-37, to make it the active window. Notice that the **Execution** window continues running. When the runforever.seq [Executing] window is the active window, the **Run** button is enabled.

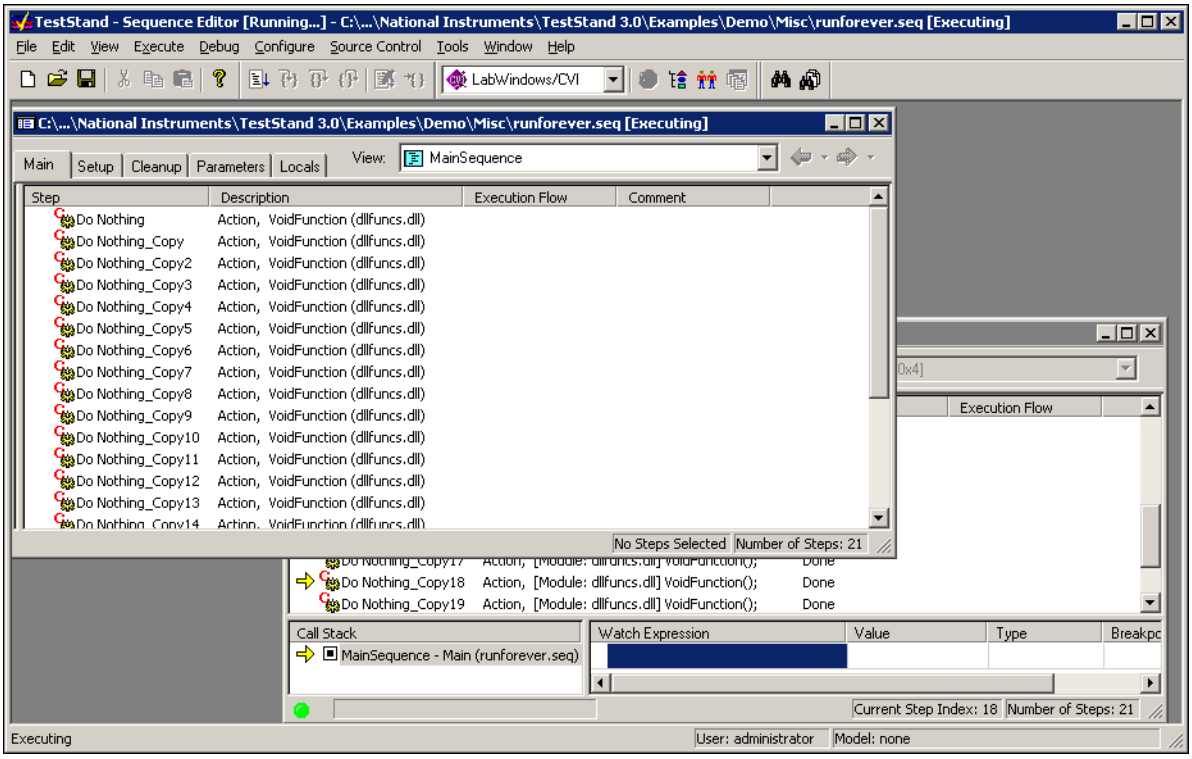


Figure 3-37. Sequence File Window

5. Click **Run** again and notice that a second Execution window opens, as shown in Figure 3-38. Two sequence executions are now running and executing independently from one another. Click **Run** several more times and notice that you can run many executions simultaneously and independently from one another.

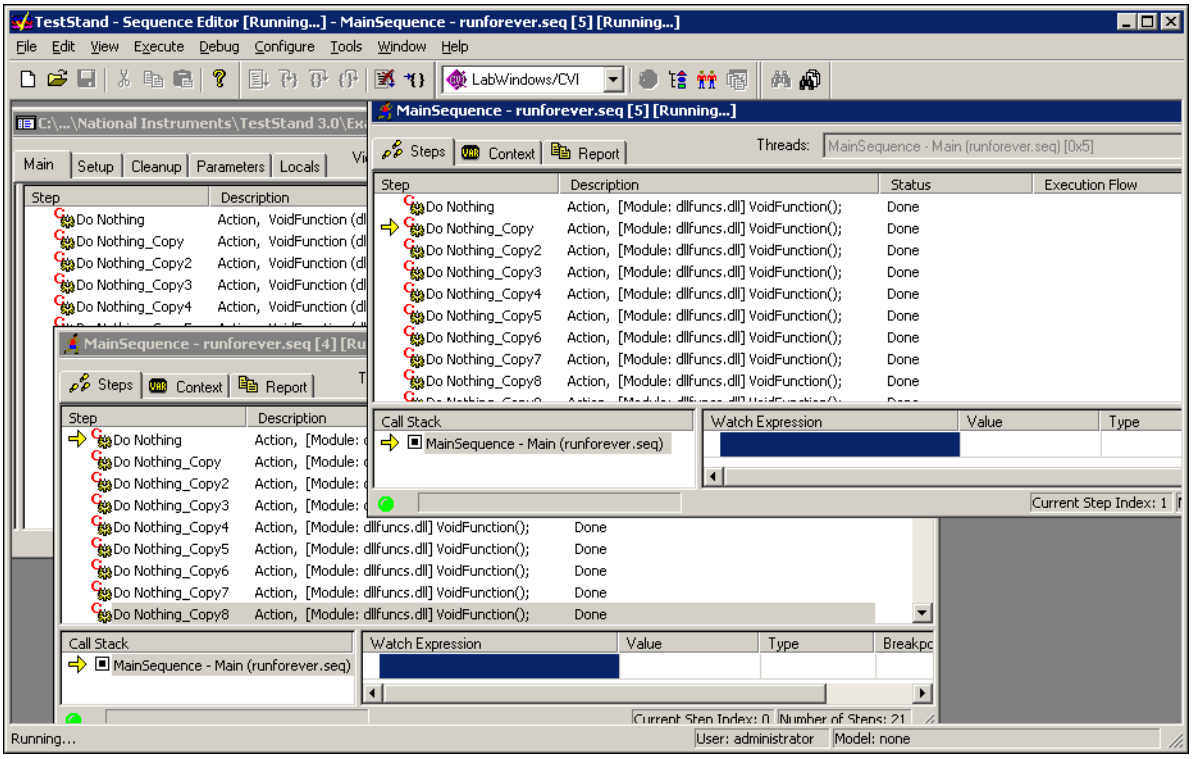


Figure 3-38. Multiple Executions Running

6. Because this sequence has no built-in way to stop itself, you can stop the sequence execution by clicking the **X** button in the top right corner of the Execution window or selecting **Debug»Terminate All**.

   When you click the **X** button, the Close Execution Window dialog box shown in Figure 3-39 launches. Select the **Terminate Execution** option to allow TestStand to halt the execution cleanly and run any cleanup steps for the executing sequence.

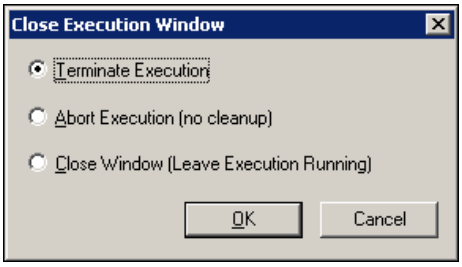


Figure 3-39. Close Execution Window Dialog Box

7. Repeat step 6 for each sequence execution.
8. When you have closed all Execution windows, close the sequence file to complete the exercise.

End of Exercise 3-4

Self Review

1. List the four actions involved in adding a step to a sequence.

2. What is the difference between an Action step and a Test step (for example, Numeric Limit Test)?

3. What is the difference between a relative path and an absolute path? Why is it better to use a relative path when configuring your code modules?

4. What is a precondition? How would you use a precondition to control execution flow within your test?

5. Which of the following methods leads to faster execution times: loading dynamically or preloading when the execution begins?

6. When you set a step to Force Fail, the code module is executed and the step status is set to Failed.True or False?

7. How would you configure a step to suspend/break if the step fails?

8. What is the behavior of the “Pass/Fail count” loop option?

9. What are pre- and post-expressions? When are the expressions evaluated?

Notes

4

TestStand Parameters, Variables, and Expressions

Lesson 4: TestStand Parameters, Variables, and Expressions

In this lesson, you will:

- **Use parameters, local variables, Sequence File Global Variables, and Station Global Variables to store and share data in your test sequence**
- **Describe the state of an executing sequence using the RunState Property Objects**
- **Use the Expression Browser dialog box to view variables and build complex expressions**

Introduction

One of the most powerful TestStand features is its flexible framework for sharing data. This lesson describes TestStand parameters and the three types of TestStand variables—local, sequence file global, and station global. The lesson also describes the RunState property object and how to utilize the TestStand Expression Browser dialog box to use variables and properties in expressions throughout the TestStand environment.

TestStand Parameters and Variables

- **Share data between steps of a sequence or among several sequences**
- **Four ways to share data collected by tests:**
  - Parameters
  - Local Variables
  - Sequence File Global Variables
  - Station Global Variables

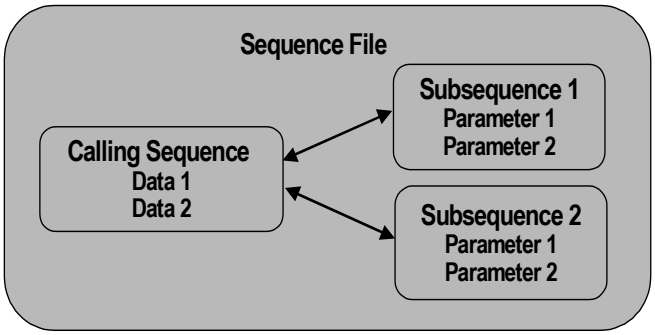
TestStand Parameters and Variables

A valuable feature of test executive software is a means to share data among tests and sequences. TestStand has four ways to share data that is collected by tests. The following four methods differ only in scope or where they can be accessed:

- **Parameters**—Pass data to a sequence when you call that sequence as a subsequence.
- **Local Variables**—Store data relevant to the execution of the sequence. Each step in the sequence can directly access sequence local 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**—Maintain statistics or represent the configuration of your test station. You can access station global variables from any sequence or step on that test station. Unlike other variables, the values of station global variables are saved from one TestStand session to the next.

TestStand Parameters

- Directly pass data between a calling sequence and the subsequence it calls
- Define in Parameters tab of Sequence File window



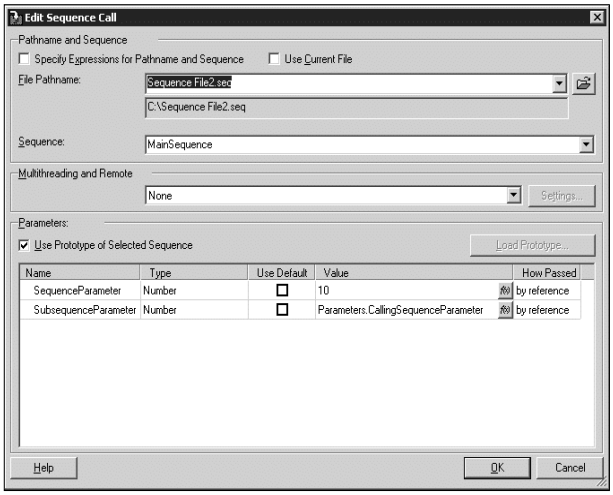
TestStand Parameters

Each sequence has its own list of parameters. Use these parameters to pass data to a sequence when you call that sequence as a subsequence. This use of parameters is analogous to passing arguments to a function call or wiring data to terminals when you call a subVI in LabVIEW. You also can specify a default value for each parameter.

You can specify the number of parameters and the data type of each parameter. You can either select a value to pass to the parameter or use the default value specified by the parameter. To create parameters in the Sequence File window, use the Parameters tab and select the desired data type. Any step within the sequence can then access the parameters.

You typically use parameters to pass data between a calling sequence and subsequence. In the calling sequence, when you select the sequence to call through the Edit Sequence Call dialog box of a Sequence Call step, you also identify the subsequence parameters and specify their values.

Parameters: Passing Data to Subsequences



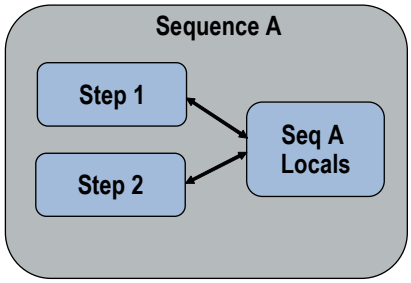
Parameters: Passing Data to Subsequences

As shown in the slide above, Sequence File2 is the sequence file and MainSequence is the sequence it calls as a subsequence. The first parameter in the subsequence is passed a constant numeric value of 10, while the second parameter is passed the value of a parameter in the calling sequence. Under the How Passed column, the by reference setting indicates that the parameter being passed is a reference to an argument, instead of a copy of its value. Passing a parameter by reference allows the calling sequence to obtain the changes that the subsequence makes to the value of the parameter.

To change how a parameter is passed, by reference or by value, right-click the parameter in the subsequence and select Pass by Reference from the context menu. A checkmark beside this option in the context menu indicates that the Pass by Reference option is enabled. If this option is not enabled, the parameter is passed by value.

Local Variables

- Share data among all steps in a sequence
- Access using expressions and/or code modules



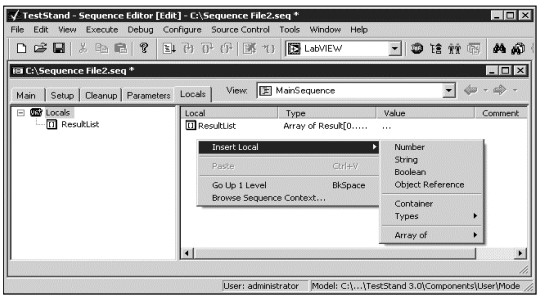
Local Variables

Local variables are shared within a single sequence. All sequence steps can get/read or set/write the value of the local variables defined for a sequence. The value of a local variable can be read or written in any TestStand expression associated with steps in that sequence or in any of the code modules called by those steps. Thus, each sequence has its own set of local variables.

For example, you can store a local variable called Serial Number at the beginning of the sequence and retrieve this serial number in subsequent tests to decide which specific tests to run.

Creating Local Variables

- **Each sequence has a Locals tab**
- **Right-click the Locals tab and select the appropriate data type to create a new local variable**



Creating Local Variables

The Locals tab of the Sequence File window shows all local variables defined for that particular sequence.

Complete the following steps to add a new local variable to a sequence.

1. Right-click inside the right pane of the Locals tab and select **Insert Local** from the context menu.
2. Select a variable type.
3. Name the variable.

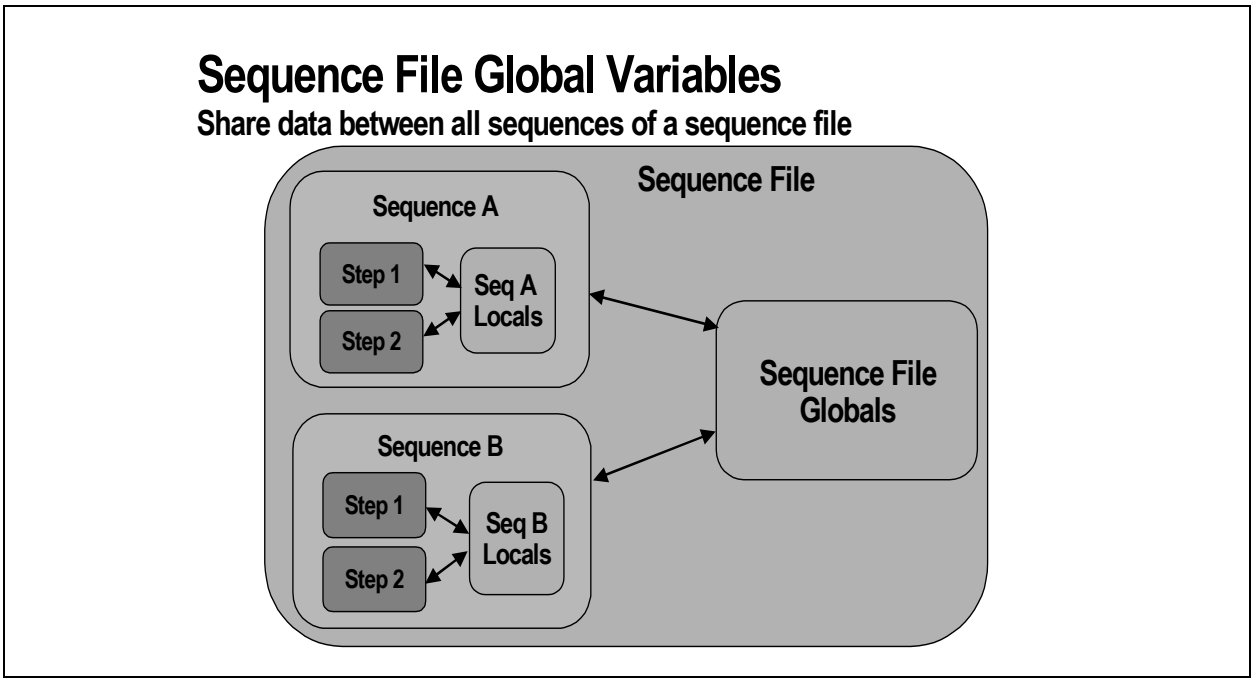
The variable types available in TestStand include number, string, Boolean, ActiveX references, and arrays of these data types. You also can create your own custom data types. Refer to Lesson 7, *Configuring TestStand*, for more information about creating custom data types.

Exercise 4-1: Using Local Variables

Objective: To insert a local variable into a sequence and pass data between steps.

Estimated Time: 20 minutes

Refer to page 4-25 for instructions for this exercise.

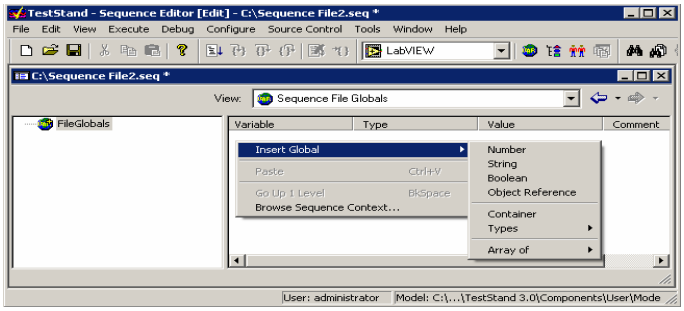


Sequence File Global Variables

Sequence file global variables are similar to local variables, but have a larger scope of usability. Besides being accessible from all steps in a given sequence like local variables, any other sequence in the sequence file can access sequence file global variables. Thus, each sequence file has its own unique set of global variables.

Creating Sequence File Global Variables

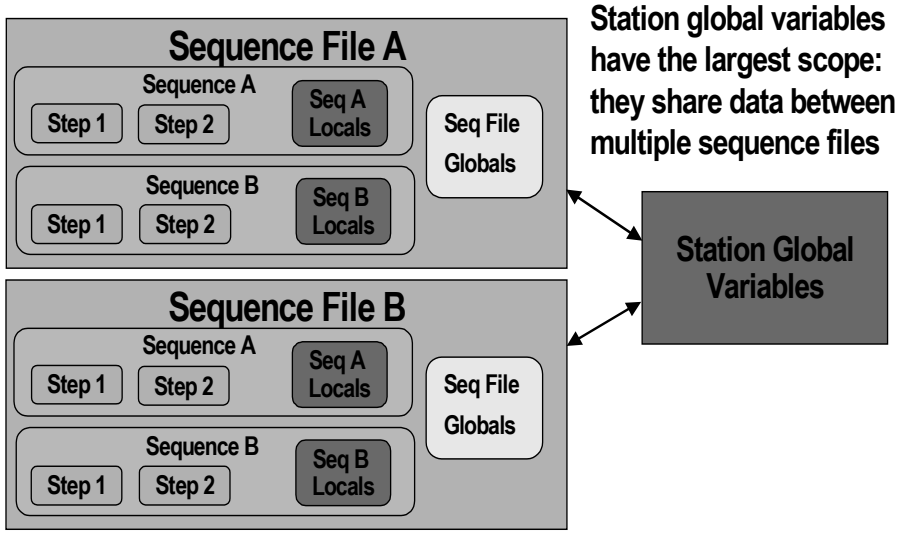
- **Select Sequence File Globals from the View ring control to display global variables**
- **To create a new global variable, right-click, choose Insert Global, and select the appropriate data type**



Creating Sequence File Global Variables

Creating a sequence file global variable is similar to creating a local variable. Complete the following steps to create a sequence file global variable:

1. Select **Sequence File Globals** from the **View** ring control of the Sequence File window to display the Sequence File Globals view, which displays all global variables defined for the sequence file.
2. Right-click the right pane of the Sequence File Globals view and select **Insert Global** from the context menu.
3. Choose the variable type to create a new sequence file global.
4. Name the new sequence file global.



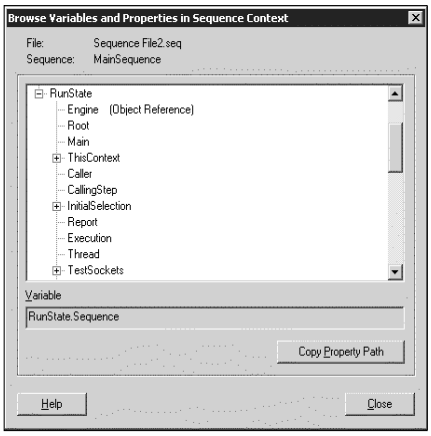
Station Global Variables

Station global variables have a larger usability scope than either local or sequence file global variables. The data contained in a station global variable is available to all sequence files. Station global variables are defined for a particular test station and their values are written to `StationGlobals.ini` each time you close the sequence editor. This enables TestStand to keep track of data values even when TestStand is shut down so users can save state information, such as the last operator to run a test.

The values of station global variables are persistent across different executions and even across different invocations of the sequence editor or operator interface. In addition, the TestStand API includes an ActiveX method that may be called to update `StationGlobals.ini` explicitly.

RunState Property Object

- Contains all the property objects that describe the state of execution in the sequence invocation
- Some of the RunState properties are shown below

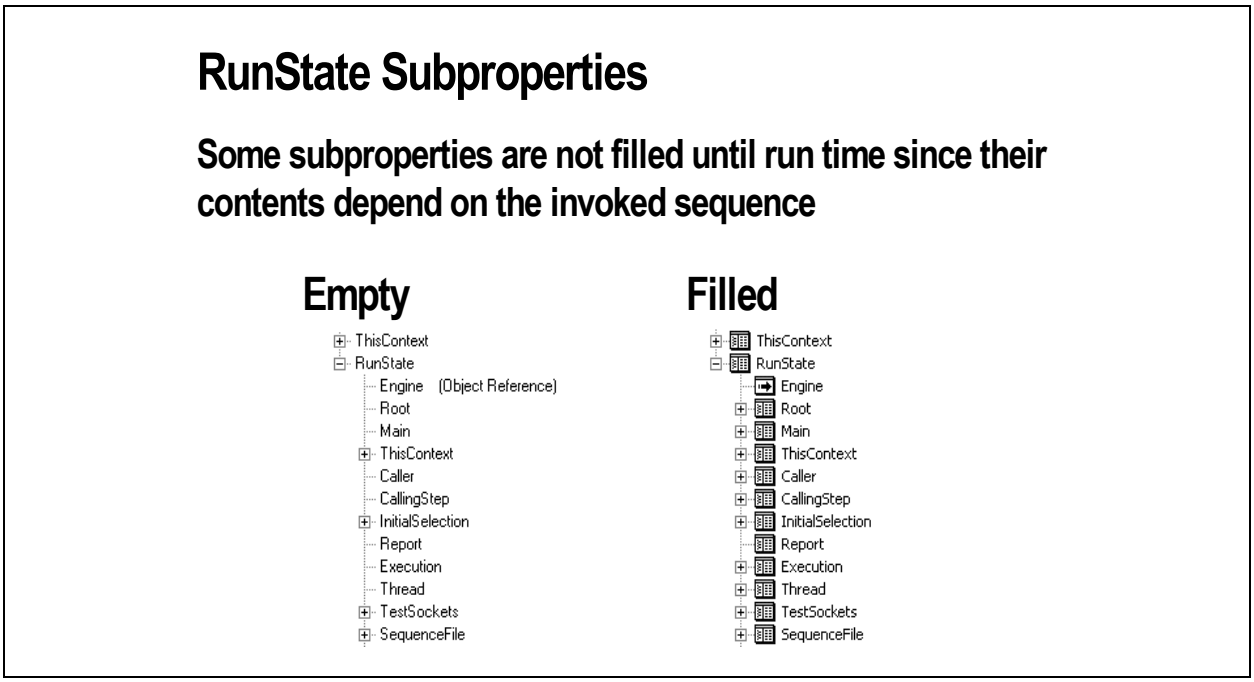


RunState Property Object

The RunState property object contains properties that describe the state of execution in the sequence invocation. The following list includes some of the properties of the RunState property object.

- **Engine**—Engine object in which the sequence invocation executes.
- **Root**—Sequence context for the root sequence invocation.
- **Main**—Sequence context for the least nested sequence that is not in a process model.
- **ThisContext**—Reference to the current sequence context.
- **Caller**—Sequence context for the sequence that called the current sequence.
- **Report**—Report object for the execution.
- **Execution**—Execution object in which the sequence invocation runs.
- **Thread**—Thread object in which the sequence invocation executes.
- **Sequence**—Run-time copy of the Sequence object for the sequence invocation.

The properties mentioned above are all objects of the TestStand API. Refer to the *TestStand II: Customization Course Manual* or the *TestStand Help* for more information about invoking methods and properties on these objects.



RunState Subproperties

Some of the properties listed under the RunState property object are empty when a sequence is not being executed. To find which subproperties are available when a sequence is not being executed, select **View»Browse Sequence Context** and expand the RunState property object. The majority of the RunState subproperties are listed at this time. However, some of the subproperties that are containers, such as the Root, Main, Caller, and Execution properties, will not contain any information.

Complete the following steps to view which RunState subproperties are available during sequence execution and what information is contained in these properties:

1. Place a breakpoint on a step within the sequence.
2. Execute the sequence.
3. When TestStand reaches the breakpoint, click the **Context** tab in the Execution window.
4. Expand the RunState property object to view the available properties according to the current sequence invocation.

Refer to the *TestStand Help* for more information about the RunState subproperties.

RunState Property Example

Complete these steps to access the UUT serial number:

1. **Set Breakpoint**
2. **Execute Sequence (Execute»Test UUTs)**
3. **Browse to the RunState property (`RunState.Root.Locals.UUT.SerialNumber`)**
4. **Copy Property Path**

RunState Property Example

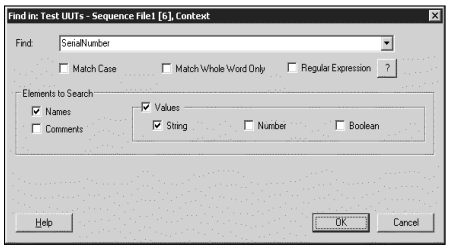
This example illustrates how to use the RunState property object to obtain the UUT serial number in a test sequence. You can access the UUT serial number from the RunState property in different ways. This example obtains the UUT serial number from the Root subproperty. Since the Root subproperty only contains information when the sequence is executing, the UUT serial number is only visible using the Root subproperty at run time. Complete the following steps to find the UUT serial number in the RunState property, then use the serial number in your test sequence.

1. Place a breakpoint at the step in your sequence where you want to access the UUT serial number.
2. Select **Execute»Test UUTs** to execute the test sequence.
3. When the execution pauses at the breakpoint, click the **Context** tab.
4. Browse to the `RunState.Root.Locals.UUT.SerialNumber` property.
5. Right-click the serial number property and select **Copy Property Path** from the context menu to copy the path to the clipboard.
6. Finish executing the sequence and return to the Sequence File window.
7. For any step that references the UUT serial number, paste the path into the **Expression** tab of the Step Properties dialog box.

RunState Property Example using Find

Complete these steps to access the UUT serial number:

1. **Set Breakpoint**
2. **Execute Sequence (Execute»Test UUTs)**
3. **Use the Find tool to locate the UUT Serial Number**
4. **Copy Property Path**



RunState Property Example using Find

You also can use the Find dialog box to locate variables, expressions, and comments within your sequence, both in development and during execution. This can be helpful when you do not know the exact path of the variable you are looking for. Complete the following steps to locate the UUT serial number using the Find dialog box:

1. Place a breakpoint at the step in your sequence where you want to access the UUT serial number.
2. Select **Execute»Test UUTs** to execute the test sequence.
3. When the execution pauses at the breakpoint, click the **Context** tab.
4. Select **Edit»Find** or press the **<Ctrl-F>** keys to open the Find dialog box.
5. Enter `SerialNumber` in the **Find** field and click the **OK** button.
6. The first result should be `RunState.Root.Locals.UUT.SerialNumber`. Double-click that entry to go to the property path where the value is located.
7. Right-click the serial number property and select **Copy Property Path** from the context menu to copy the path to the clipboard.
8. Finish executing the sequence and return to the Sequence File window.
9. For any step that references the UUT serial number, paste the path into the **Expression** tab of the Step Properties dialog box.

Expression Browser

- Simplifies building complex expressions
- Click the Expression Browse button next to any TestStand control to access the Expression Browser dialog box
- Includes two tabs:
  - Variables/Properties
  - Operators/Functions

Expression Browser

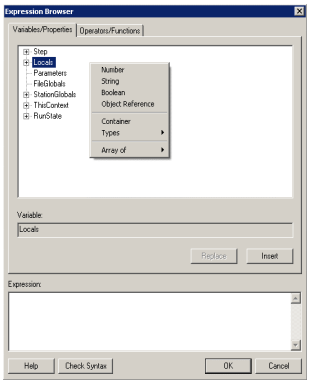
The TestStand Expression Browser dialog box allows you to use the TestStand variables you create within the sequence editor.

An expression is a formula that calculates a new value from the values of multiple variables, properties, and expression functions. You can use TestStand expressions to evaluate a value dynamically instead of hard coding a value for a simple variable or property value. Throughout the sequence editor, there exists locations or input controls where you can create a custom expression for evaluation purposes during sequence execution. Some of these locations include the Post Actions, Loop Options, and Expressions tabs. You can also create custom expressions within the Statement, Limit Loader, and Message Popup steps.

You can add an expression in any TestStand control that has an **Expression Browse** button beside it. Click **Expression Browse** to launch the Expression Browser dialog box. You can launch the Variable Browser by selecting **View»Browse Sequence Context**. This browser is similar to the Expression Browser dialog box, but does not include the Functions/Operators tab. Although it is not necessary to use the Expression Browser dialog box to enter an expression, it is useful for locating variables and functions when building complex statements and avoids typographical errors in these expressions.

Defining Variables and Parameters in the Expression Browser Dialog Box

The Expression Browser dialog box allows you to add and remove variables and parameters in your test sequences



Defining Variables and Parameters

Using the Expression Browser Dialog Box

The Expression Browser dialog box includes the following tabs:

- **Variables/Properties**—Displays all variables and properties accessible to the current context.
- **Operators/Functions**—Includes all valid TestStand operators and functions.

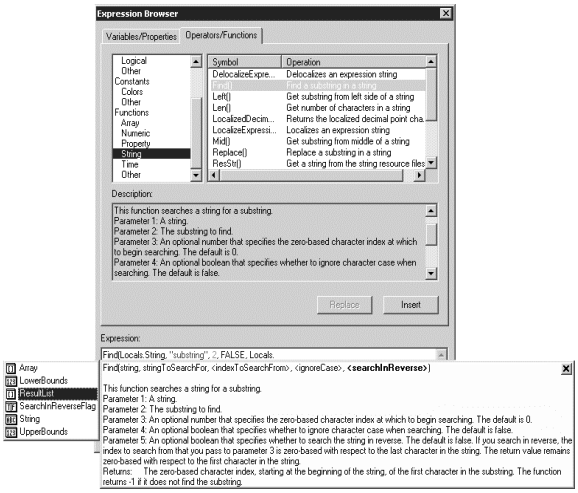
Using these tabs in conjunction allows you to build complex expressions quickly and easily. The Expression Browser dialog box also includes a Check Syntax button that determines whether the expression syntax is correct. You can then insert the expression you created into the current window from which you launched the Expression Browser dialog box.

Defining Variables and Parameters

When you create expressions in the Expression Browser dialog box, you might determine that you need an additional variable or parameter that you have not previously defined in the sequence editor. You can define the new variable or parameter in the current tab or window of the Expression Browser dialog box. You can declare new parameters, local, file global, and station global variables from within the Expression Browser dialog box. When you right-click the variable type or parameter in the Expression Browser dialog box, a context menu appears that allows you to select the data type of the new variable or parameter you want to define. This context menu offers the same declaration choices as the other variable editing views

in the sequence editor. Once you have defined the variable or parameter within the Expression Browser dialog box, it appears in the respective tab or window within the sequence editor as if you had defined it there originally.

Operators/Functions Tab



Operators/Functions Tab

The Expression Browser dialog box provides syntax highlighting similar to the syntax highlighting in most modern ADEs such as Microsoft Visual Studio, and highlights function parameters by data type. By default, the colors are the same as the colors in LabVIEW: purple strings, green Boolean values, and orange fractional numbers.

TestStand automatically completes lookup strings you enter in the Edit Expression control. For example, if you enter `Locals`, all the local variables defined in the sequence display in a drop-down menu. You can then select one of the menu items and continue typing to automatically complete the lookup string.

When you use one of the TestStand defined functions located on the Operators/Functions tab, a function tip displays to provide help about the parameters and return values of the function.

The following terms are useful as you develop more complex TestStand sequences using the Expression Browser dialog box:

- **SequenceContext**—Contains references to all global variables, local variables, and step properties in active sequences. The sequence context contents change depending on the currently executing sequence and step.
- **ThisContext**—Refers to the current sequence context. You usually use this property to pass the entire sequence context as an argument to

another sequence or a code module. Sequence contexts are ActiveX automation reference handles you can use with code modules.

- **RunState**—Contains properties that describe the state of execution in the sequence invocation.

Exercise 4-2: Examining the Uses and Differences Between Local, File Global, and Station Global Variables

Objective: To create local, file global, and station global variables and use the Expression Browser dialog box to understand the scope of these variables.

Estimated Time: 15 minutes

Refer to page 4-29 for instructions for this exercise.

Exercise 4-3: Using Expressions to Change Step Properties

Objective: To modify the limit properties of a Numeric Limit step type.

Estimated Time: 25 minutes

Refer to page 4-36 for instructions for this exercise.

Exercise 4-4: Passing Parameters

Objective: To use expressions and parameters in subsequence executions.

Estimated Time: 25 minutes

Refer to page 4-44 for instructions for this exercise.

Exercise 4-5: Dynamically Selecting Which Sequence to Run (Challenge)

Objective: To create a test that dynamically selects which sequence to run.

Estimated Time: 20 minutes

Refer to page 4-56 for instructions for this exercise.

Lesson 4 Summary: Parameters, Variables, and Expressions

- **TestStand has different scopes for data sharing**
  - Parameters share data between calling sequences and subsequences
  - Local variables share data between steps in a sequence
  - Global variables share data between all sequences within a sequence file
  - Station global variables share data between all sequence files within the TestStand environment
- **RunState properties describe the state of execution**
- **The Expression Browser dialog box is a tool for building expressions. You also can use it to create variables and parameters**

Summary

In this lesson, you learned how to share data within the TestStand environment. Each method for sharing data serves a particular purpose. You may want to think about which situations would require the use of parameters, local, sequence file global, or station global variables. For example, as you begin deployment to the production floor, what information on the test station is needed across shifts? If you have a DLL, should all the data manipulation be internalized or shared using TestStand local variables?

As you access and modify variables, you also might want to make use of the RunState properties to obtain information such as the UUT serial number, the UUT loop index, and the next step index. Using multiple variables and properties in your expression can create complex statements. TestStand eases this task by allowing you to arbitrarily create complex expressions through the Expression Browser dialog box. It also improves programming efficiency by allowing you to create variables and parameters from within the Expression Browser dialog box as well as providing tools including automatic completion, syntax highlighting, and function tips.

Exercise 4-1 Using Local Variables

Objective: To insert a local variable into a sequence and pass data between steps.

Local variables share data between steps in the same sequence. This type of variable has the smallest scope, but is used most often. This exercise demonstrates how to use local variables to change the frequency of the StartBeep step, configured in Exercise 3-1, *Creating Steps in the Sequence Editor*. This is done by adding a local variable to the sequence.

1. In the sequence editor, open the `Ex 4-1 Local Variables.seq` sequence file located in the `C:\Exercises\TestStand I` directory.

   This file is the same as the solution for Exercise 3-1. The solutions for the exercises are located in the `C:\Solutions\TestStand I` directory.

2. Click the **Locals** tab.
3. Right-click the right pane of the Locals tab and select **Insert Local» Number** from the context menu, as shown in Figure 4-1.

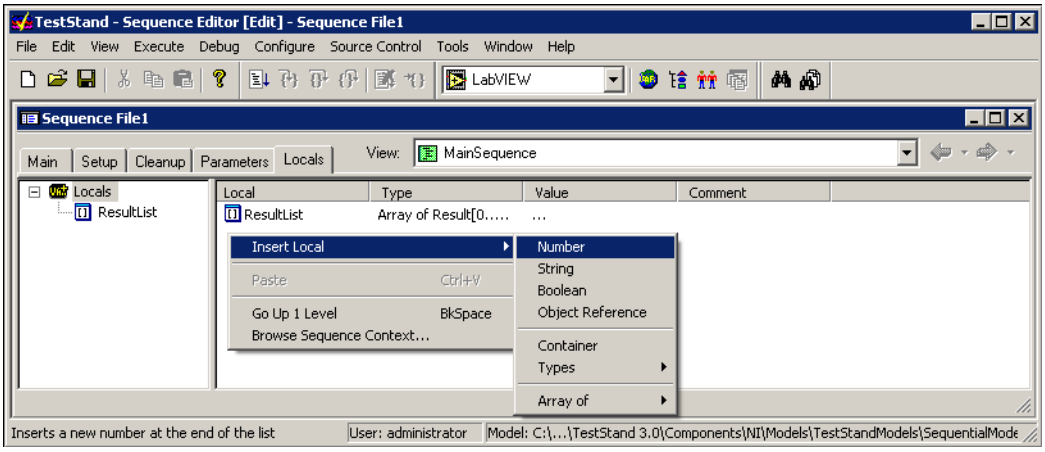


Figure 4-1. Inserting a Local Variable

4. Name the local variable `Frequency`. This variable sets the audible alarm tone when the ROM test fails.
5. Click the **Main** tab and change the module called during the `ROM Test` step by right-clicking the step and selecting **Specify Module** from the context menu.
6. In the Edit LabVIEW VI Call dialog box, click the **File Browse** button and select `ROM Test - Local Variable.vi` located in the `C:\Exercises\TestStand I\VIs` directory.

   If TestStand prompts you to resolve the path to the VI, select the **Use a relative path for the file you selected** option. If TestStand prompts you to keep the current parameter values, click **Yes**.

7. In the Edit LabVIEW VI Call dialog box that opens, enable the **Show VI Front Panel when Called** option as shown in Figure 4-2.

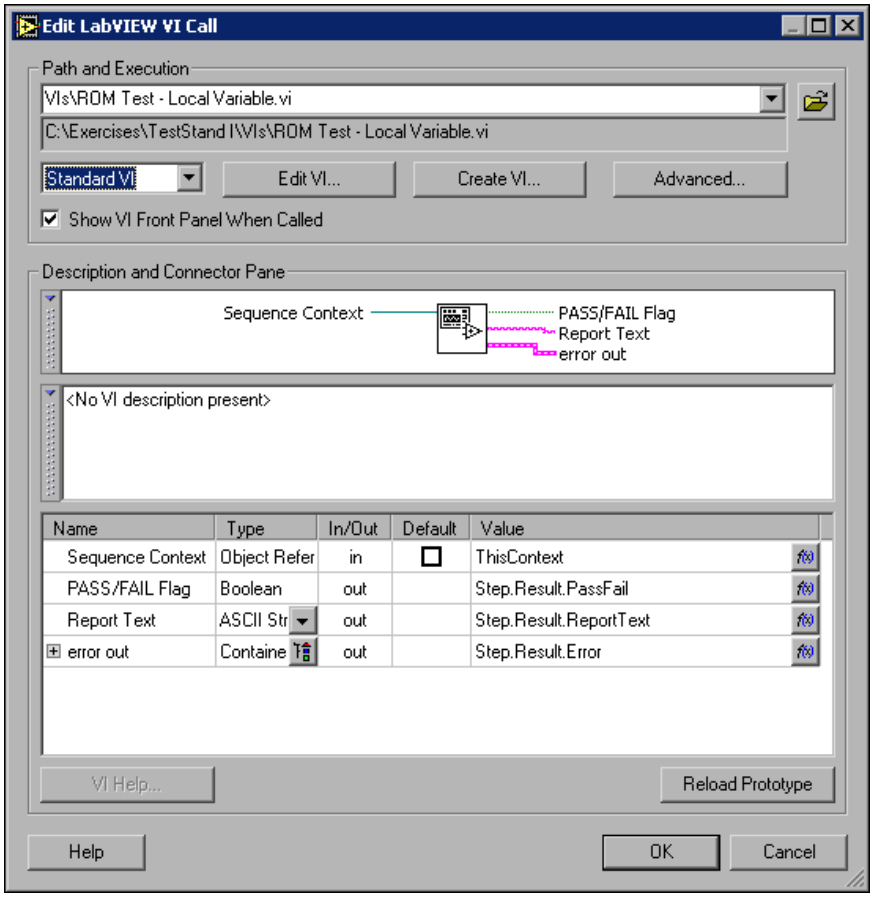


Figure 4-2. Edit LabVIEW VI Call Dialog Box

Caution Failing to select the **Show VI Front Panel When Called** option can cause an infinite loop to run in the VI.

If you are familiar with National Instruments LabVIEW programming, continue to step 8. Otherwise, skip to step 11.

8. Click **Edit VI** in the Edit LabVIEW VI Call dialog box.

   Notice that the front panel has a frequency control for entering the frequency value.

9. Select **Window»Show Block Diagram** to open the block diagram.

   Notice that the TestStand - Set Property Value VI passes the numeric frequency value to TestStand. Refer to the *LabVIEW Help* for more information about TestStand VIs and their functions.

10. Close the VI.

11. Click **OK** to close the Edit LabVIEW VI Call dialog box.

12. Right-click the `AlarmStart` step and select **Specify Module** from the context menu to open the Edit C/C++ DLL Call dialog box.
13. Complete the following steps to set the `AlarmStart` step to use the `Frequency` local variable:
    a. In the **Parameters** section, select the **Value Expression** column for the parameter tone. If this paramterer already has an expression, delete it.
    b. Click **Expression Browse** and select `Locals.Frequency` from the Expression Browser dialog box.
    c. Click **Insert**, then click **OK**.

    The Edit C/C++ DLL Call dialog box should now have Locals.Frequency set as the value for the parameter tone as shown in Figure 4-3.

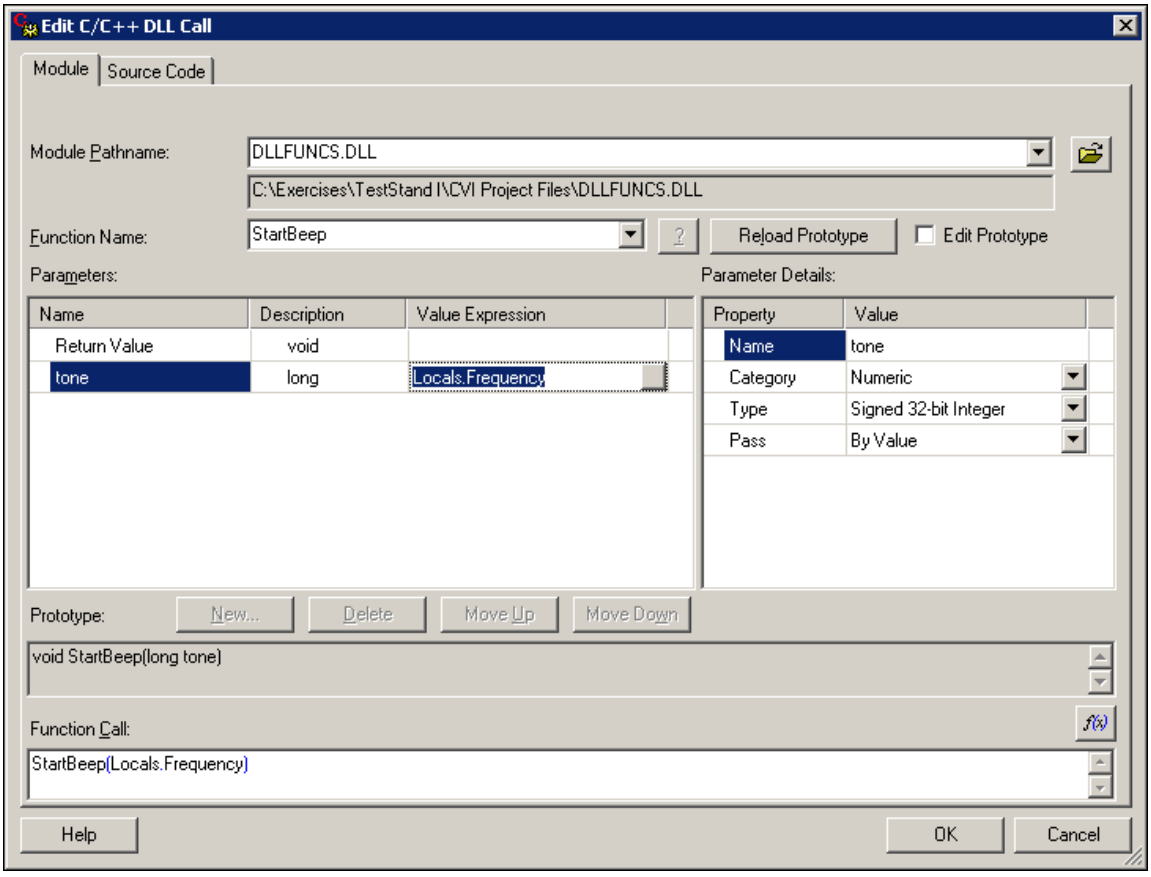


Figure 4-3. Edit DLL Call Dialog Box

14. Click **OK** to return to the Sequence File window.
15. Save the sequence file.
16. Select **Execute»Test UUTs** and try varying the frequency for several different tests. Remember that the ROM test must be set to `Fail` for the alarm to sound. The alarm sounds a tone that is proportional to the frequency.

End of Exercise 4-1

Exercise 4-2 Examining the Uses and Differences between Local, File Global, and Station Global Variables

Objective: To create local, file global, and station global variables and use the Expression Browser dialog box to understand the scope of these variables.

It is important to understand the three different types of variables in TestStand and their differences.

- Local variables share data between steps in the same sequence.
- File global variables share data between a step in one sequence and a step in another sequence in the same sequence file.
- Station global variables share data between steps in two sequences in different sequence files.

This exercise demonstrates how to create each type of variable and emphasizes the differences between them.

Part A: Local Variables

1. Select **File»New Sequence File** in the sequence editor.
2. Click the **Locals** tab.

   A local variable shares information between different steps in a single sequence.
3. Right-click in the right pane of the Locals tab and select **Insert Local» Number** from the context menu to create a numeric local variable.
4. Name the local variable `LocalA`.
5. Select **View»Browse Sequence Context** to launch the Browse Variables and Properties in Sequence Context dialog box. The variable browser displays the variables and properties in the TestStand environment relative to the current sequence.

   Additional Information
   The Browse Variables and Properties in Sequence Context dialog box is different from the Expression Browser dialog box because it does not include the Operators/Functions tab. Any location that can accept an expression includes an Expression Browse button that launches the Expression Browser dialog box. The Expression Browser dialog box displays the hierarchy of properties and variables for the sequence context from which it is invoked. The Expression Browser dialog box becomes more useful when more advanced topics are discussed.
6. Click the **+** sign to expand the list of local variables and notice that the `LocalA` variable created in the previous step is listed, as shown in Figure 4-1.

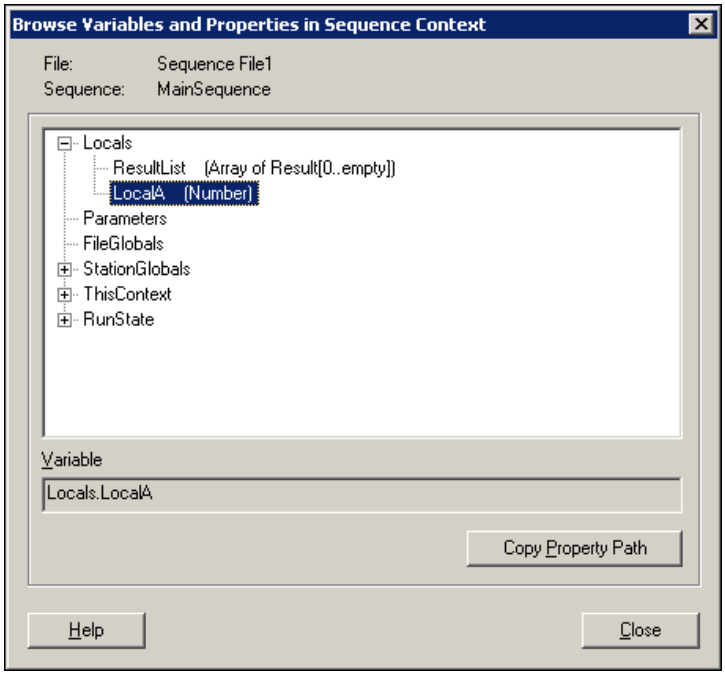


Figure 4-1. Browse Variables in Sequence Context Dialog Box

7. Click **Close** to return to the Sequence File window and continue with the exercise.
8. Select **File»Save As** and save the sequence file as `Ex 4-2 All Variables1.seq` in the `C:\Exercises\TestStand I` directory.
9. Select **File»New Sequence File** and repeat steps 2 and 3 to create another local variable.
10. Name this variable `LocalB`. Figure 4-2 shows the Locals tab.

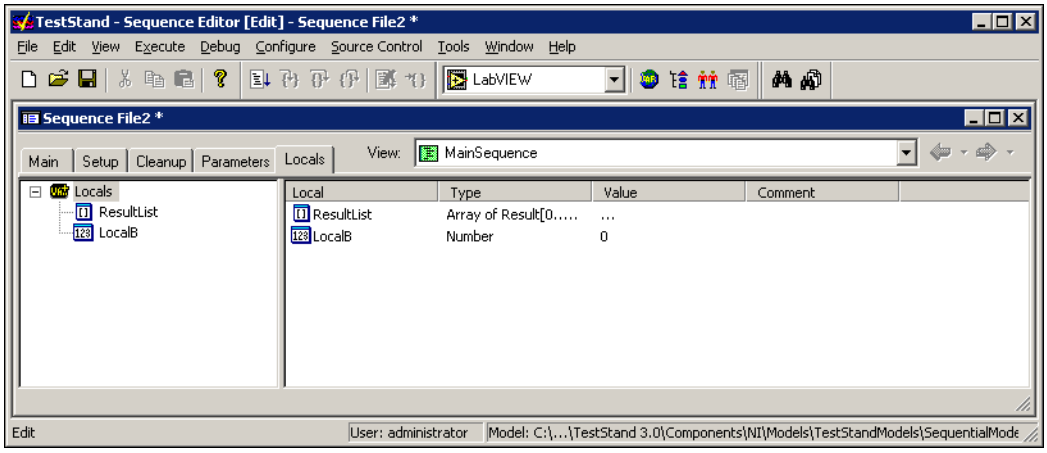


Figure 4-2. Locals Tab in Sequence File Window

11. Select **View»Browse Sequence Context** to launch the Browse Variables and Properties in Sequence Context dialog box.

    Only the local variable created in this sequence is visible because the variable LocalA, created in the first sequence, is not accessible. The scope of local variables is limited to a single sequence; they can be accessed only from within the sequence in which they were created.

12. Click **Close** to exit the Browse Variables and Properties in Sequence Context dialog box.

13. Select **All Sequences** from the **View** ring control.

    Notice that there is only one sequence, named `MainSequence`, in this sequence file.

14. Right-click in the All Sequences view and select **Insert Sequence** from the context menu to create another sequence.

15. Name this sequence `TestSequence`. Figure 4-3 shows the final result.

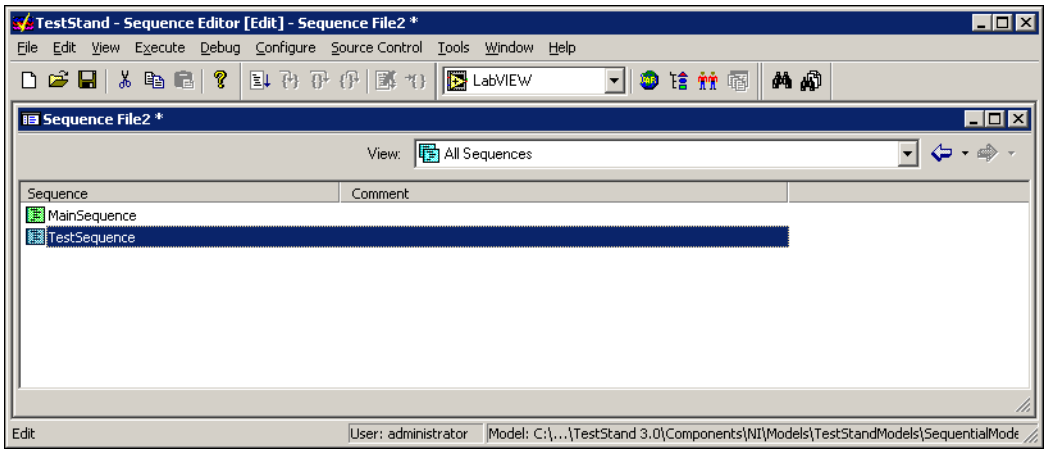


Figure 4-3. All Sequences View

16. Select **TestSequence** from the **View** ring control to view the contents of this sequence. The sequence should be empty.

17. Select **View»Browse Sequence Context** to launch the Browse Variables and Properties in Sequence Context dialog box.

    Notice that LocalB is not available for use in this sequence, as shown in Figure 4-4, because LocalB was defined in the `MainSequence` and not in the `TestSequence`.

    Even though the two sequences are in the same sequence file, the local variables are not shared between them.

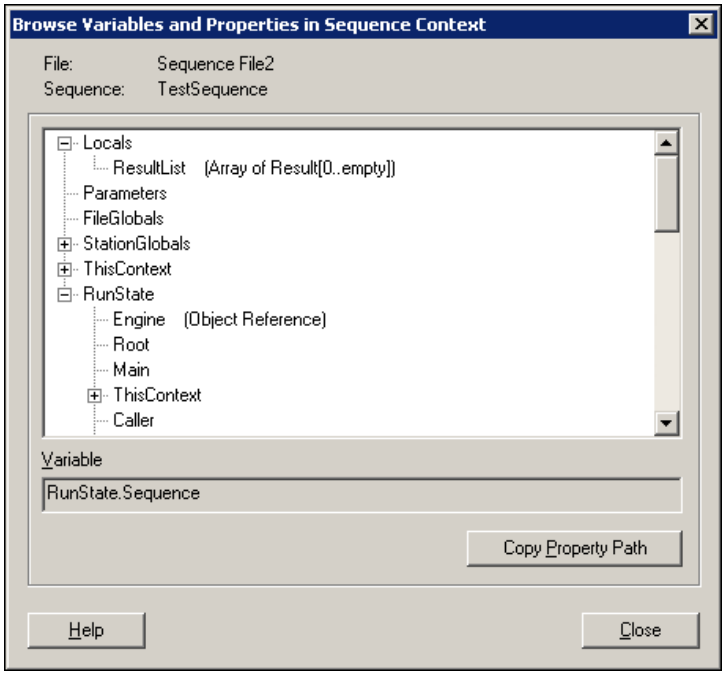


Figure 4-4. Browse Variables and Properties in Sequence Context Dialog Box

18. Click **Close** to exit the Browse Variables and Properties in Sequence Context dialog box.
19. Select **File»SaveAs** and save the sequence file as `Ex 4-2 All Variables2.seq` in the `C:\Exercises\TestStand I` directory.

    Leave this sequence file open for the next part of this exercise.

End of Part A

Part B: Sequence File Global Variables

A sequence file global variable passes data between different sequences in the same sequence file.

1. Select **Sequence File Globals** from the **View** ring control to open the Sequence File Globals view.
2. Right-click in the right pane of the File Globals tab and select **Insert Global»Number** from the context menu.
3. Name this variable `TestFileGlobalA`.
4. Save the sequence file.
5. Select **TestSequence** from the **View** ring control to view the contents of this sequence.
6. Select **View»Browse Sequence Context** to launch the Browse Variables and Properties in Sequence Context dialog box.

Notice that the `TestFileGlobalA` variable is available for use in this sequence, as shown in Figure 4-5.

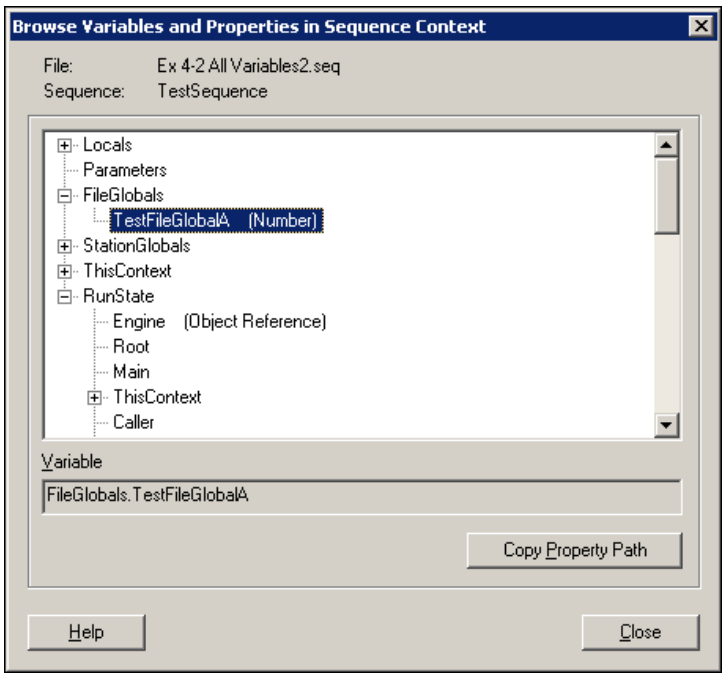


Figure 4-5. Browse Variables and Properties in Sequence Context Dialog Box

7. Click **Close** to return to the sequence editor.
8. Select **MainSequence** from the **View** ring control to open the MainSequence.
9. Select **View»Browse Sequence Context** and notice that the `TestFileGlobalA` variable also is available for use in this sequence.

   The data in the sequence file global variable you created in the `MainSequence` is available for use in all other sequences in the `Ex 4-2 All Variables2.seq` file. In this way, you can use sequence file global variables to pass data among different sequences in the same sequence file.
10. Click **Close** to return to the sequence editor.

End of Part B

Part C: Station Global Variables

Station global variables share data between different sequence files. In this part of the exercise, you will create a station global variable and examine its scope.

1. Click **Station Globals**, shown at left, on the toolbar to open the Station Globals window.

2. Right-click in the right pane of the Station Globals window and select **Insert Global»Number** from the context menu, as shown in Figure 4-6.

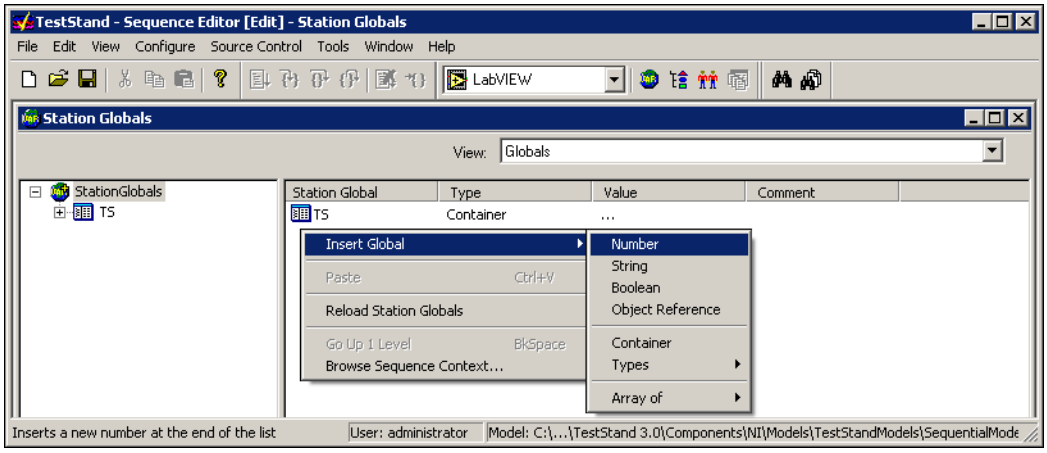


Figure 4-6. Inserting a Station Global Variable

3. Name the station global variable `StationGlobal1`.
4. Select **Window»Ex 4-2 All Variables2.seq**. Next, select **View»Browse Sequence Context** to open the Browse Variables and Properties in Sequence Context dialog box.
5. Click the **+** sign next to **Station Globals** and notice that `StationGlobal1` is available for use.
6. Click **Close** to return to the Sequence File window.
7. Save and close the `Ex 4-2 All Variables2.seq` sequence file.
8. Open the `Ex 4-2 All Variables1.seq` file if it is not already open and select **View»Browse Sequence Context** to open the Browse Variables and Properties in Sequence Context dialog box.
9. Click the **+** sign next to **Station Globals** and notice that `StationGlobal1` is available for use.

   Any sequence you execute on this test station can use station global variables. In addition, station global variables are persistent; that is, they keep their value in a `.ini` file from one TestStand session to the next.
10. Click the **+** sign next to the **TS** station global variable and notice that this container holds a value called `LastUserName`.

    This station global variable stores the name of the last user that logged into TestStand. Because LastUserName is stored in a station global variable, when the sequence editor is launched, the user name from the previous session displays in the Login dialog box.

11. Click **Close** to return to the Sequence File window.
12. Close the sequence file, save any changes.
13. Click the **X** at the top right corner of the Station Globals window to close it and continue to the next exercise.

End of Part C

End of Exercise 4-2

Exercise 4-3 Using Expressions to Change Step Properties

Objective: To modify the limit properties of a Numeric Limit step type.

This exercise demonstrates how to programmatically modify the properties of steps . You use the Property Loader step type to load limits and properties from files and databases. However, for illustrative purposes, this exercise uses TestStand expressions to change limits manually. Refer to Lesson 6, *Importing and Exporting Properties*, for more information about the Property Loader step type.

Part A: Modifying Step Properties Using Pre-Expressions

1. Open the `Ex 4-3 Using Expressions to Change Step Properties.seq` file located in the `C:\Exercises\TestStand I` directory.
2. Run the sequence several times, selecting different tests to fail.
3. Click **Stop** in the UUT Information dialog box to stop the sequence.

   Notice that the Video test always fails. This step fails because the returned measurement is outside the specified limits of 15 and 19, as shown in Figure 4-1.

ROM Test	
Status:	Passed
Module Time:	0.0002537
RAM Test	
Status:	Passed
Module Time:	0.0002551
Video Test	
Status:	Failed
Measurement:	5
Limits:	
Low:	15
High:	19
Comparison Type:	GTLT (> <)
Module Time:	0.0002556

Figure 4-1. TestStand Report

Additional Information

The measurement and limits are properties of each step. Different types of steps have different properties. For example, Numeric Limit steps such as the Video test have measurement, high limit, and low limit properties. As the sequence executes, the value of some of these properties changes depending on what is passed back from code modules.

4. Return to the Sequence File window, right-click the `Video Test` step, and select **Edit Limits** from the context menu.

   Examine the Edit Numeric Limit Test dialog box as shown in Figure 4-2, but do not change the limit settings. You use the Edit Numeric Limit Test dialog box to set the comparison type and input limits you want to use in the comparison.

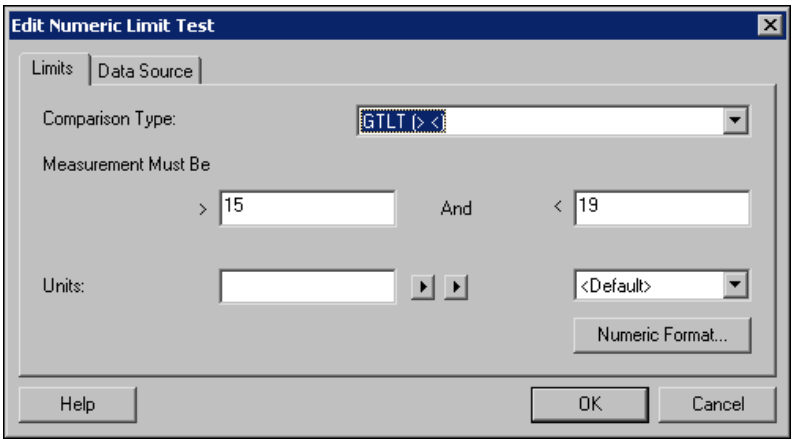


Figure 4-2. Edit Numeric Limit Test Dialog Box

5. Click **OK** to close the Edit Numeric Limit Test dialog box.
6. Double-click the `Video Test` step to open the Step Properties dialog box and click the **Expressions** tab.

   The Expressions tab allows you to modify expressions before and after the step executes.
7. Place the cursor in the **Pre-Expression** text box and click **Expression Browse** next to the text box to open the Expression Browser dialog box.
8. Click the **+** sign next to `Step` to view all the Video Test step properties. Next, click the **+** sign next to `Limits` to display the High and Low limit properties.
9. Select the `Low` property and click **Insert**. The Expression Browser dialog box appears, as shown in Figure 4-3.

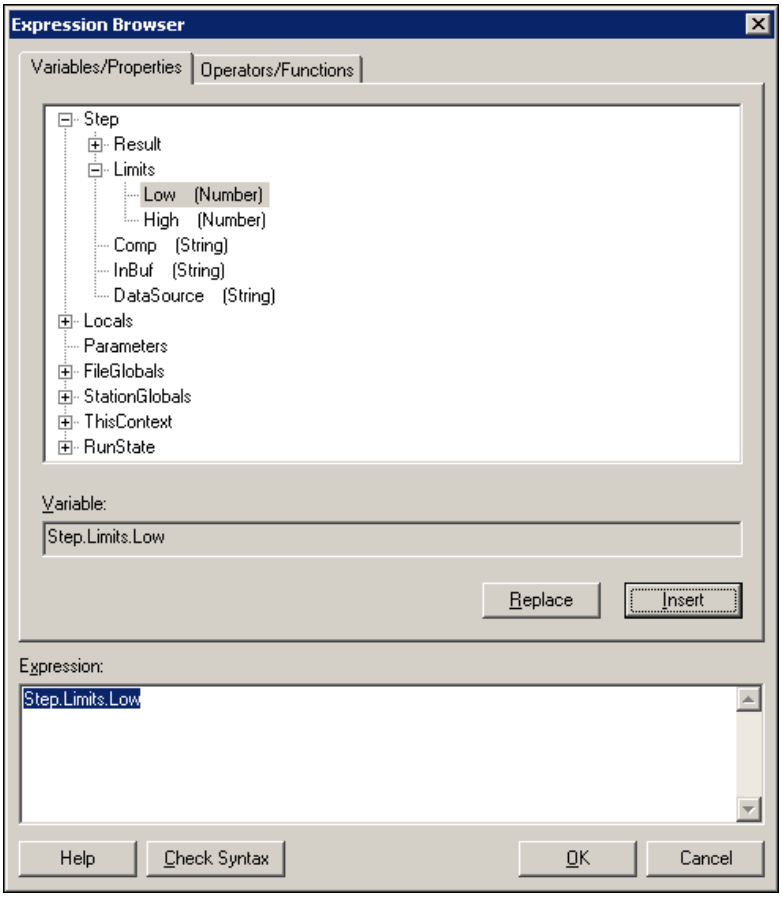


Figure 4-3. Expression Browser Dialog Box

10. Place the cursor after `Step.Limits.Low` and enter `= 2`, to set the low limit to a value of 2.

Tip Use a comma to separate multiple expressions. Include a comma in the expression box after the first expression.

11. Select the `High` property and click **Insert**. Place the cursor after `Step.Limits.High` and enter `= 12` to set the high limit. The completed Expression Browser dialog box is shown in Figure 4-4.

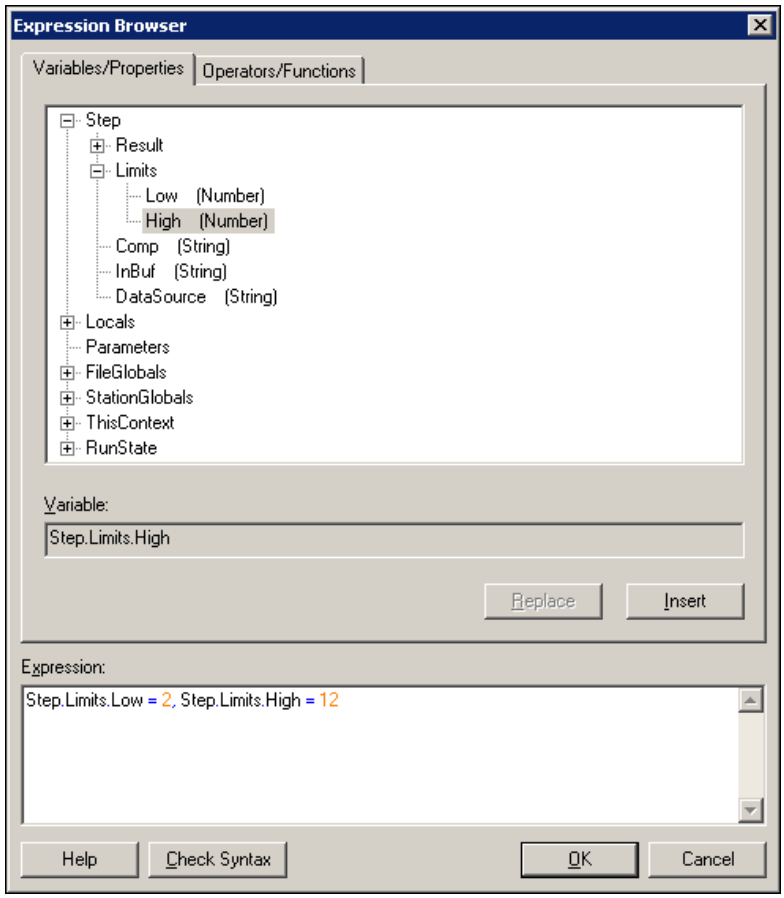


Figure 4-4. Expression Browser Dialog Box

12. Click **OK** to close the Expression Browser dialog box.
13. TestStand inserts this expression into the **Pre-Expression** text box of the Step Properties dialog box. Verify that it appears as shown in Figure 4-5.

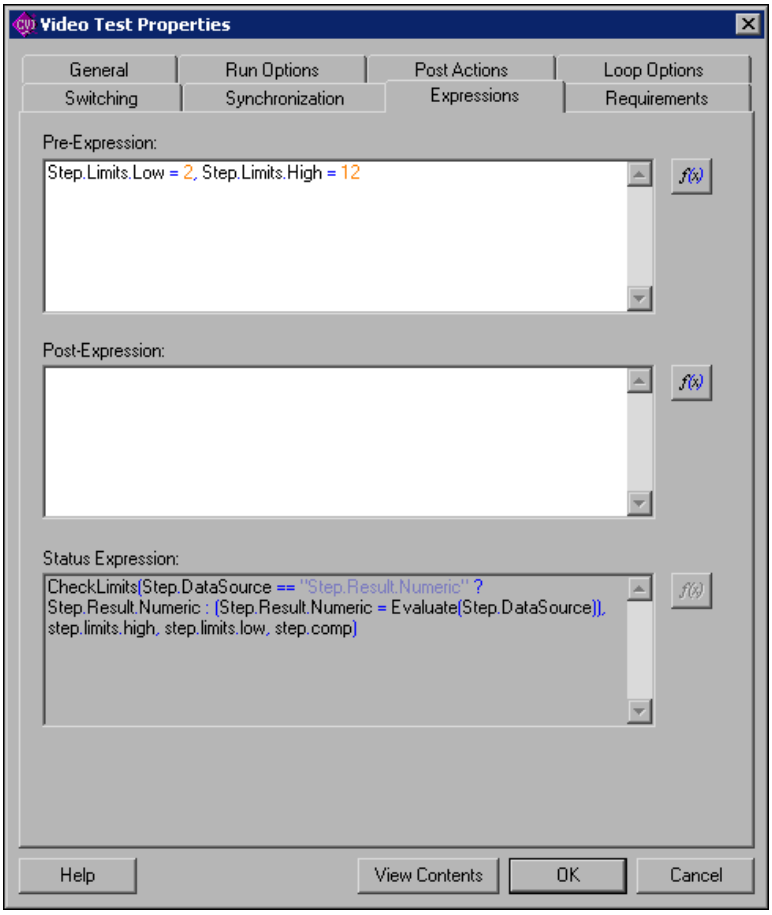


Figure 4-5. Pre-Expression Added to Step Properties Dialog Box

14. Click **OK** to return to the Sequence File window.
15. Save the sequence file.
16. Run this sequence using Single Pass again and notice that the Video Test step passes this time because the limits are now set to `2` and `12`.

End of Part A

Part B: Modifying Step Properties from Another Step

This section demonstrates how to change step properties from another step by modifying the Video Test step limits.

1. Double-click the `Video Test` step and click the **Expressions** tab in the Step Properties dialog box.
2. Delete the entry `Step.Limits.Low = 2, Step.Limits.High = 12` from the **Pre-Expression** control that you created in Part A.
3. Click **OK** to return to the Sequence File window. Run this sequence to ensure that it fails as it did before.

4. Right-click the `RAM Test` step and select **Insert Step»Statement** from the context menu to insert a new Statement step. Statement steps are useful for manipulating expressions.
5. Name the step `Modify Video Limits`.
6. Right-click the `Modify Video Limits` step and select **Edit Expression** from the context menu to open the Edit Statement Step dialog box.
7. Click **Expression Browse** in the Edit Statement Step dialog box. Browse for `RunState.Sequence.Main["Video Test"].Limits.Low` and click **Insert**.
8. In the Convert Step Names to Unique IDs dialog box that opens, click **No**. This dialog box allows you to insert step names using their unique IDs, allowing you to have the same step name in a sequence. However, you use the actual step name in this exercise.
9. Place the cursor after `RunState.Sequence.Main["Video Test"].Limits.Low` in the **Expression** text box and enter `= 2`, to set the low limit, as shown in Figure 4-6.

Tip Use a comma to separate multiple expressions. Be sure to add a comma in the Expression text box after the first expression.

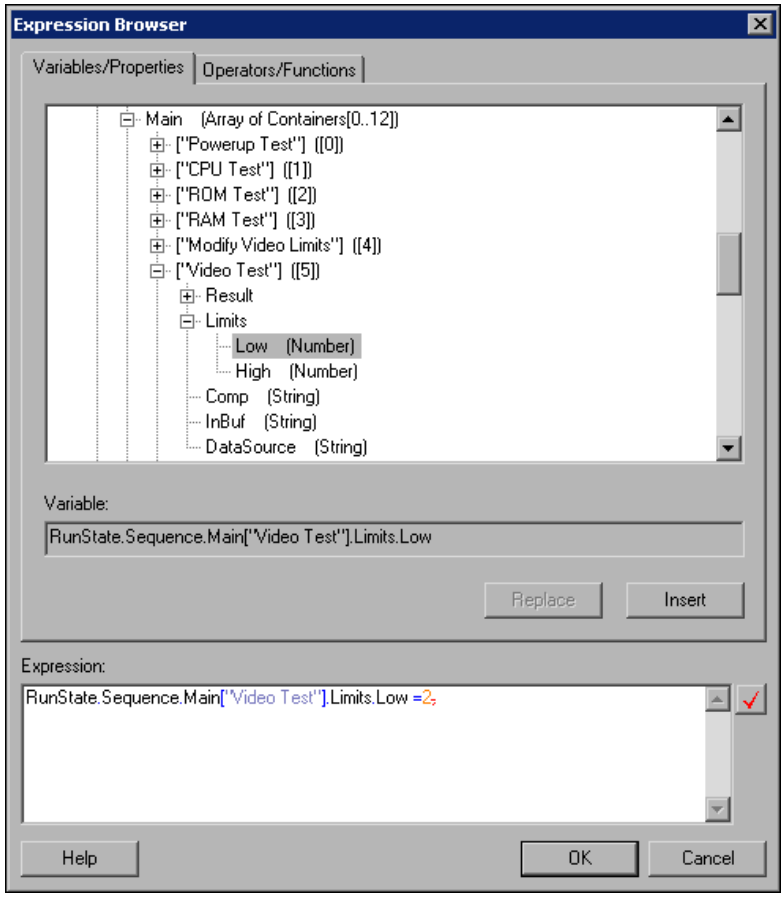


Figure 4-6. Expression Browser

10. Browse to `RunState.Sequence.Main["Video Test"].Limits.High` and click **Insert** to add this expression to the **Expression** text box. In the Convert Step Names to Unique IDs dialog box that appears, click **No**. Place the cursor at the end of the line and enter `= 12`.

11. Click **Check Expression for Errors** to make sure that there are no syntax errors and then click **OK** twice to return to the Edit Statement Step dialog box. Figure 4-7 shows the completed Edit Statement Step dialog box.

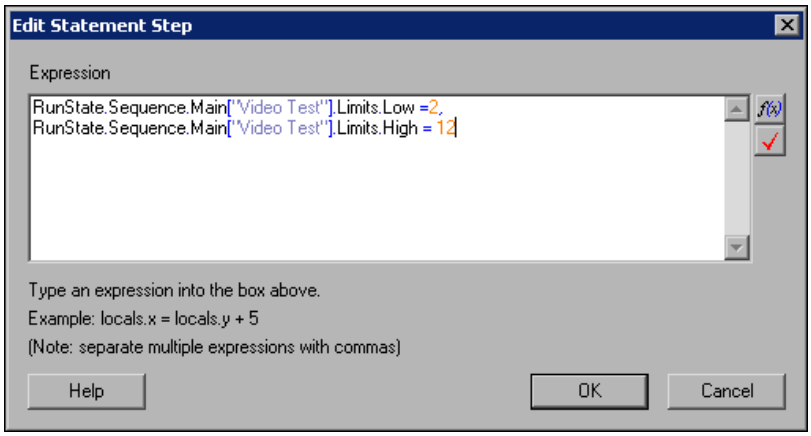


Figure 4-7. Edit Statement Step Dialog Box

12. Click **OK** to return to the Sequence File window.
13. Run the sequence file using Single Pass to make sure that it works correctly.
14. Save and close the sequence file when you finish.

End of Part B

End of Exercise 4-3

Exercise 4-4 Passing Parameters

Objective: To use expressions and parameters in subsequence executions.

This exercise demonstrates the following concepts:

- Using parameters to pass information among calling sequences and subsequences.
- Developing a sequence that executes subsequences according to user inputs at run time.

The sequence you create contains steps that prompt an operator for a CPU type and the number of CPUs to test. TestStand then uses the CPU type information during the sequence execution to execute one of two different subsequences, and passes the number of CPUs to the subsequence as a parameter to set the number of times to repeat the test.

Part A: Adding Steps to the Sequence

1. Open the `Ex 4-4 Passing Parameters.seq` file located in the `C:\Exercises\TestStand I` directory.
2. Before continuing with the exercise, click the **Parameters** tab. Select the **AMDProcessor** sequence from the **View** ring control and observe the sequence paramers. Now select the **INTELProcessor** sequence from the **View** ring control and observe the parameters.

   Notice that the AMDProcessor sequence has two parameters, and the INTELProcessor sequence has three parameters. This exercise describes how to share data with these subsequences using parameters and how TestStand uses default values if no data is specified for a given parameter.
3. Select **MainSequence** from the **View** ring control to return to the `MainSequence`.
4. Select the **Main** tab to open the main step group.
5. Right-click the `Powerup Test` step and select **Insert Step»Message Popup** from the context menu.
6. Rename the new step `Select CPU Type`.
7. Right-click the `Select CPU Type` step and select **Edit Message Settings** from the context menu.
8. Change the following control values in the Configure Message Popup Step dialog box:

Title Expression: `"Select CPU"`

Message Expression: `"Please select the CPU Type for the UUT"`

Button 1: `"INTEL CPU"`

Button 2: `"AMD CPU"`

Cancel Button: `None`

Figure 4-8 shows the completed Configure Message Popup Step dialog box.

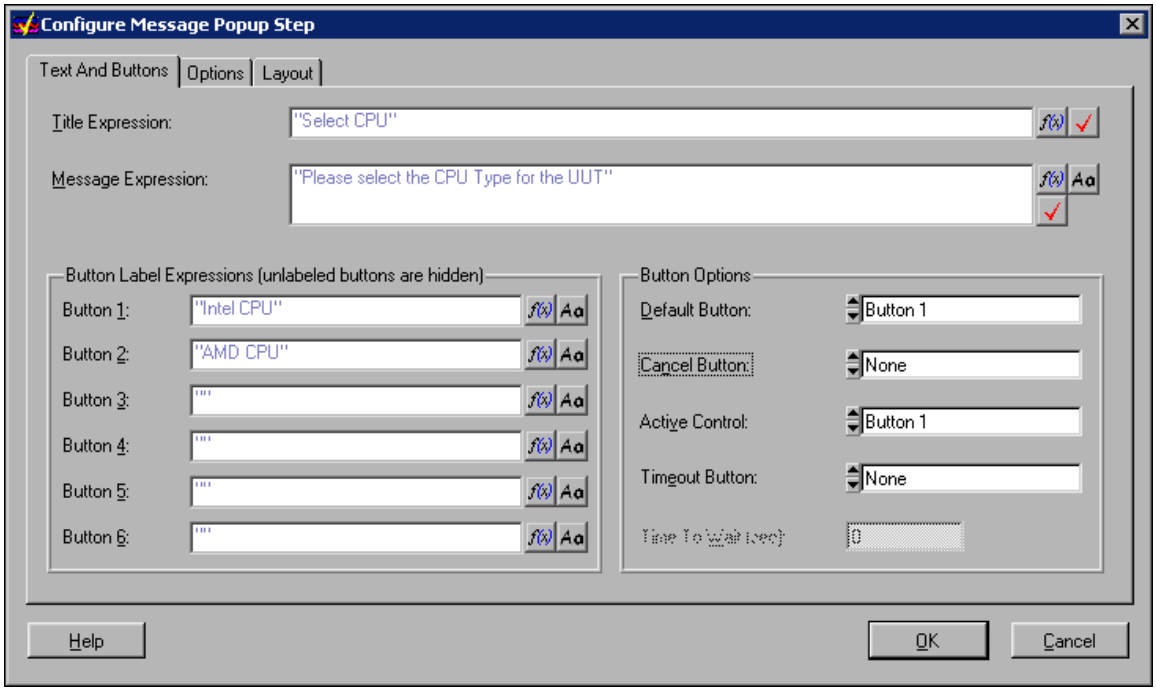


Figure 4-8. Configure Message Popup Step Dialog Box

9. Click **OK** to close the Configure Message Popup Step dialog box.
10. Right-click the `Select CPU Type` step and select **Properties** from the context menu to open the Step Properties dialog box for this step.
11. Click the **Expressions** tab.
12. Click **Expression Browse** next to the **Post-Expression** text box to open the Expression Browser dialog box. In the Variables/Properties tab, right-click **Locals** and select **Insert String** from the context menu, as shown in Figure 4-9.

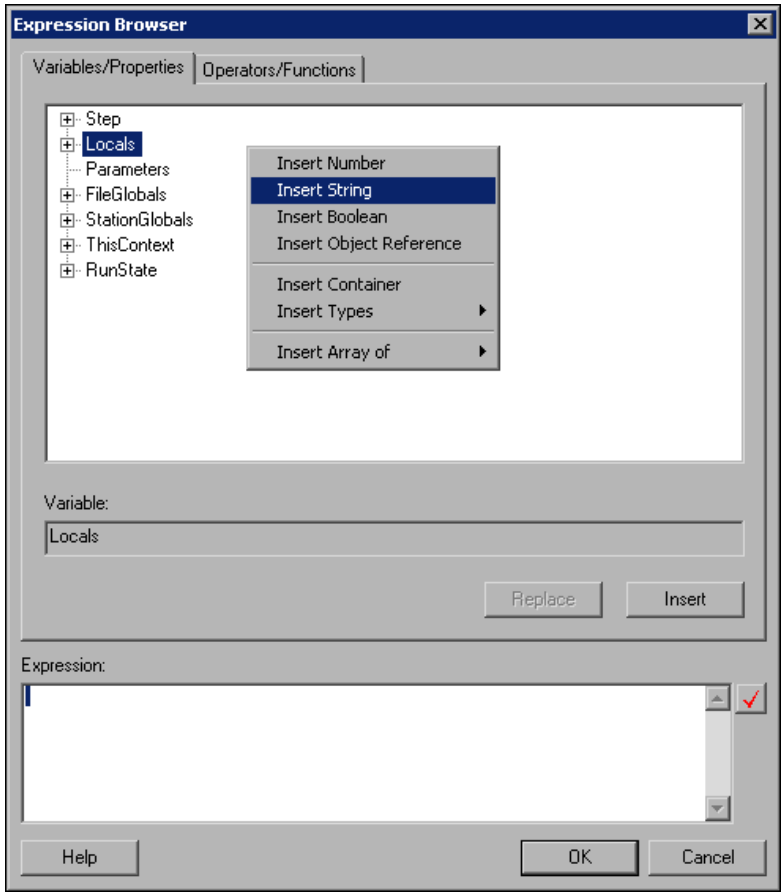


Figure 4-9. Inserting a String in the Expression Browser Dialog Box

13. Name the string `CPUType` and press the **<Enter>** key. Click **Insert** to begin the expression.
14. Enter the remainder of the expression in the **Expression** text box by using the Expression Browser dialog box or entering the following text:

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

    This expression assigns the string value "AMD" or "INTEL" to the local variable, depending on which button the user clicks.
15. Click **OK** to close the Expression Browser. Figure 4-10 shows the completed Step Properties dialog box.

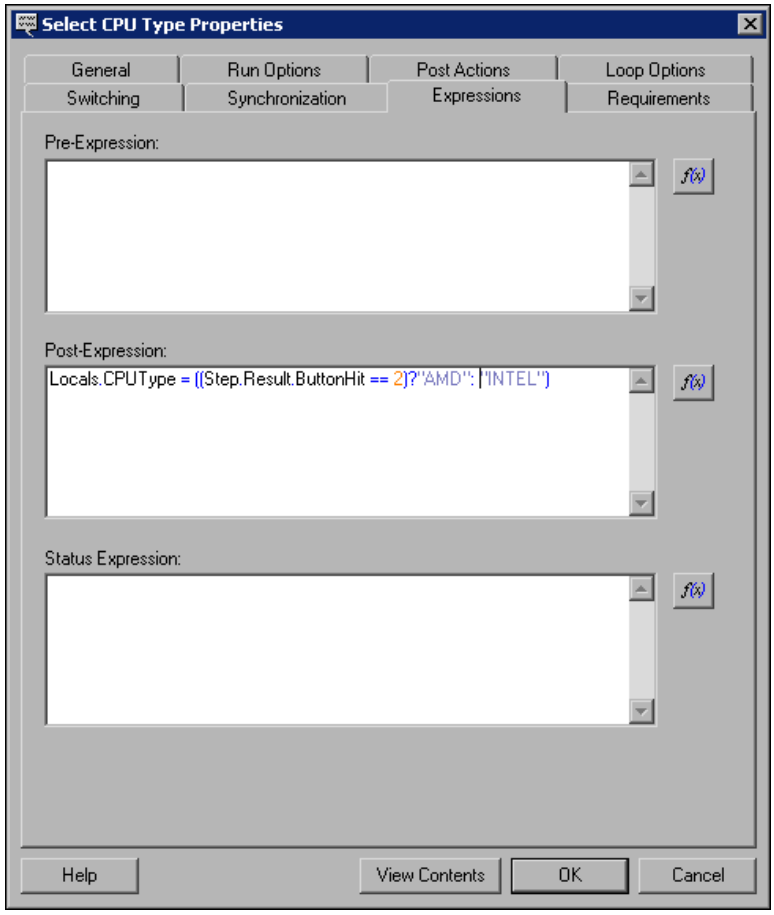


Figure 4-10. Completed Step Properties Dialog Box

16. Click **OK** to close the Step Properties dialog box.
17. Right-click the `Select CPU Type` step and select **Insert Step» Message Popup** from the context menu.
18. Rename the new step `Specify Number of CPUs`.
19. Right-click the `Specify Number of CPUs` step and select **Edit Message Settings** from the context menu.
20. Change the following control values on the Configure Message Popup Step dialog box:

Title Expression: `"Specify Number of CPUs"`

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`

Figure 4-11 shows the completed Configure Message Popup Step dialog box.

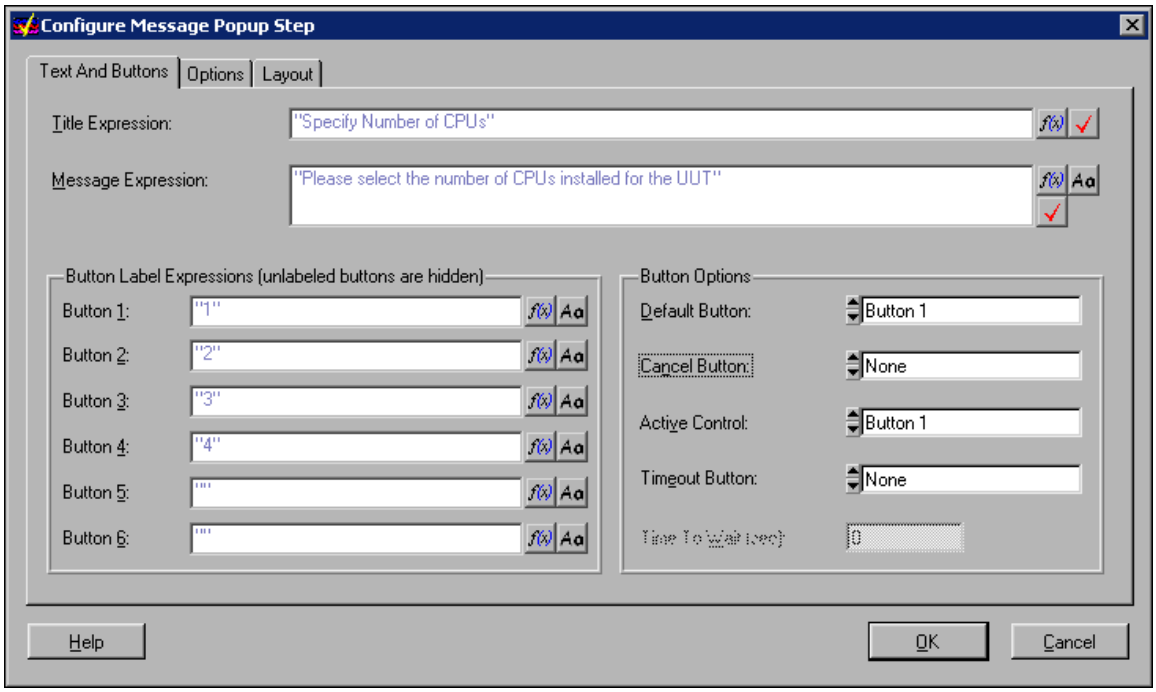


Figure 4-11. Completed Configure Message Popup Step Dialog Box

21. Click **OK** button to close the Configure Message Popup Step dialog box.
22. Right-click the `Specify Number of CPUs` step and select **Insert Step»Sequence Call** from the context menu.
23. Name the Sequence Call step `CPU Test`.
24. Right-click the `CPU Test` step and select **Specify Module** from the context menu.
25. Enable the Specify Expressions for Pathname and Sequence option.

    Additional Information

    This option allows an expression to determine the name of the sequence file to call. By default, TestStand expects the user to enter the name and path of the sequence file. Enabling this option causes TestStand to evaluate an expression to determine the path and filename for the sequence to call at this step.
26. Enable the **Use Current File** option.

Additional Information

This option specifies the current sequence file as the location for the subsequence call to search. Because both subsequences that can be called are in the same sequence file as `MainSequence`, you should select this option. The only thing that needs to be selected is the name of the sequence to be called within the sequence file.

27. Enter the following value in the **Sequence** text box:

    ```
    Locals.CPUType + "Processor"
    ```

28. Click **Load Prototype** and select the prototype for the Sequence Call step.

    Additional Information

    The Load Prototype button allows TestStand to look at the sequence file to call and determine whether any parameters must be passed into the sequence. If there are parameters, using this option allows TestStand to determine the number of parameters and their names. Exercise 3-3 demonstrated how to create the parameter manually to pass into the subsequence call. This exercise uses the prototype to save time and prevent typing mistakes.

29. Disable the Use Current File option in the Load Sequence Prototype dialog box.

30. Click **File Browse** and open the `Ex 4-4 Passing Parameters.seq` file located in the `C:\Exercises\TestStand I` directory. This specifies the file and path for the sequence prototype.

31. In the Load Sequence Prototype dialog box, select **AMDProcessor** sequence from the **Sequence** ring control.

    Figure 4-12 shows the completed Load Sequence Prototype dialog box.

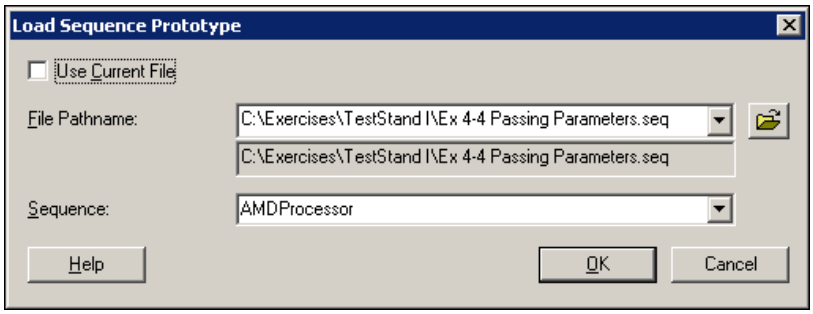


Figure 4-12. Load Sequence Prototype Dialog Box

32. Click **OK** to close the Load Sequence Prototype dialog box.

    Additional Information

    Notice that TestStand populates the Parameters section of the Edit Sequence Call dialog box with the parameter list for the sequence. This sequence requires two parameters—CPUsInstalled and CPUFail.

33. Place the cursor in the Value column of the CPUsInstalled parameter.
34. Click **Expression Browse** and use the Expression Browser dialog box to create the following expression:

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

    Click **OK** to close the Expression Browser dialog box.

    Figure 4-13 shows the completed Edit Sequence Call dialog box.

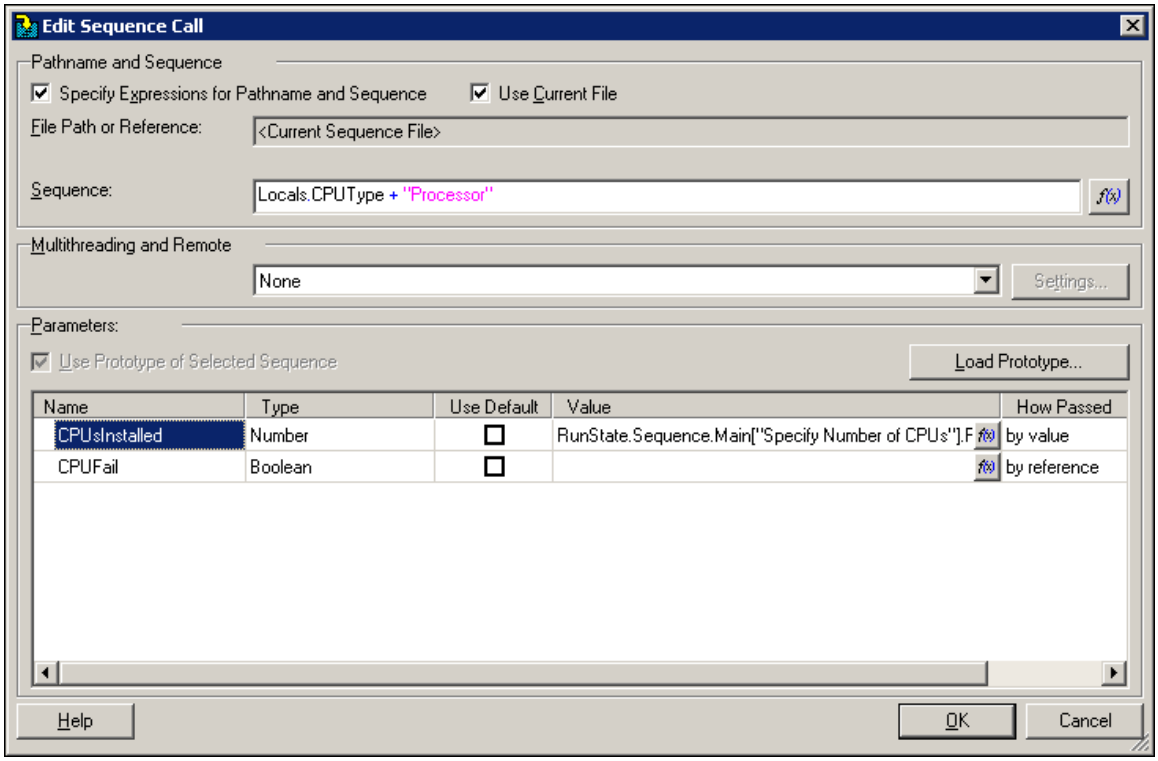


Figure 4-13. Edit Sequence Dialog Box

35. Place the cursor in the Value column of the CPUFail parameter.
36. Click **Expression Browse** and use the Expression Browser dialog box to create the following expression:

    ```
    Locals.CPUFail
    ```

37. Click **OK** to close the Expression Browser dialog box.

    Figure 4-14 shows the completed Edit Sequence Call dialog box.

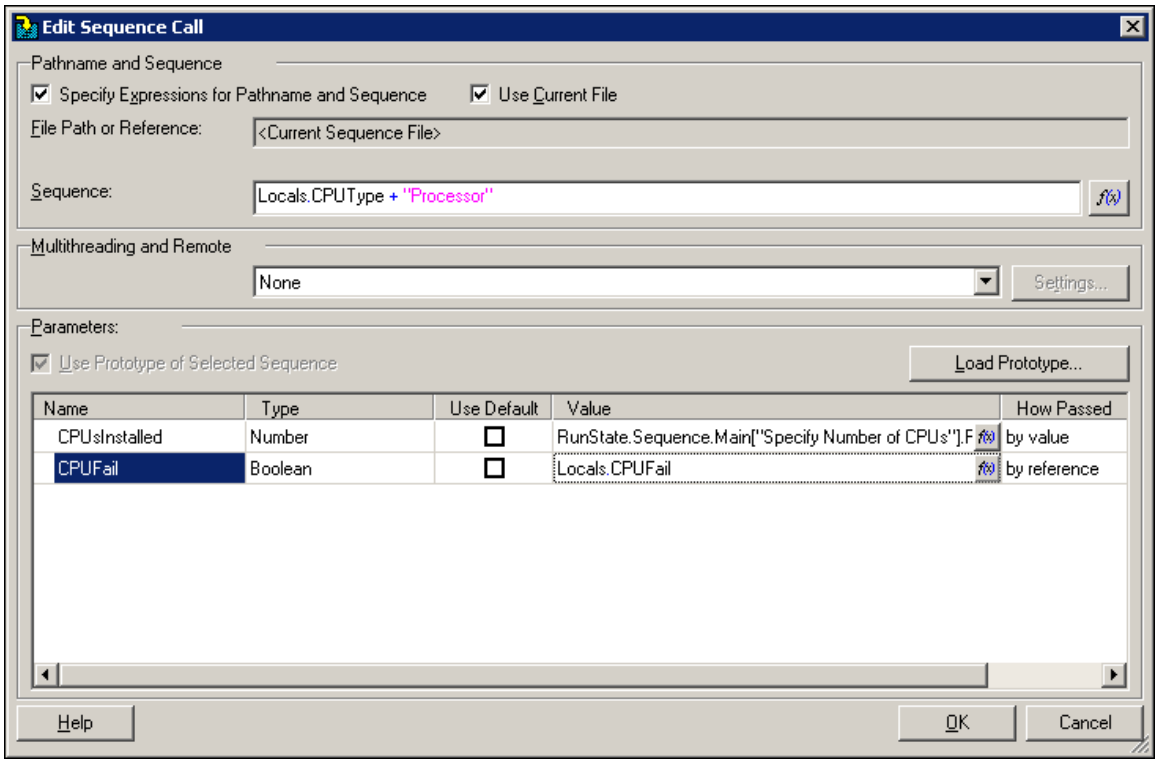


Figure 4-14. Completed Edit Sequence Call Dialog Box

38. Click **OK** to close the Edit Sequence Call dialog box. Figure 4-15 shows the completed Sequence File window.

    Additional Information

    You now have assigned the values to pass to the subsequence.
    The sequence now knows which subsequence to call and what values to pass to the subsequence as parameters. Because the values are passed using parameters, there is no need to create sequence file or station global variables.

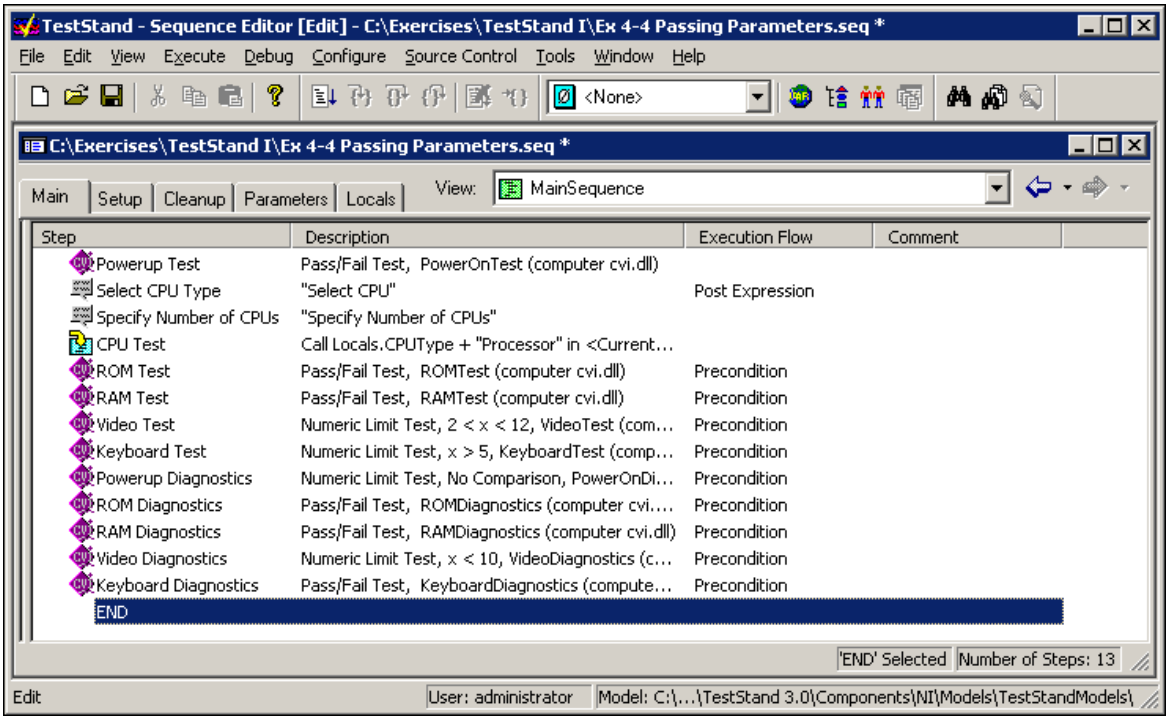


Figure 4-15. Completed Sequence File Window

39. Save the sequence.

End of Part A

Part B: Running a Sequence

Complete the following steps to run the sequence you created in Part A.

1. Click to the left of the `CPU Test` step and place a breakpoint or right-click the step and select **Breakpoint»Toggle Breakpoint** from the context menu.
2. Select **Execute»Single Pass** to run the sequence.
3. Click **Done** in the Test Simulator dialog box.
4. Click **INTEL CPU** in the Select CPU dialog box
5. Click **2** in the Number of CPUs dialog box.
6. When the execution pauses at the `CPU Test` step, single-step into the subsequence by selecting **Debug»Step Into**.

   Notice that the Call Stack pane lists the `INTELProcessor` sequence at the bottom of the sequence call stack, as shown in Figure 4-16.

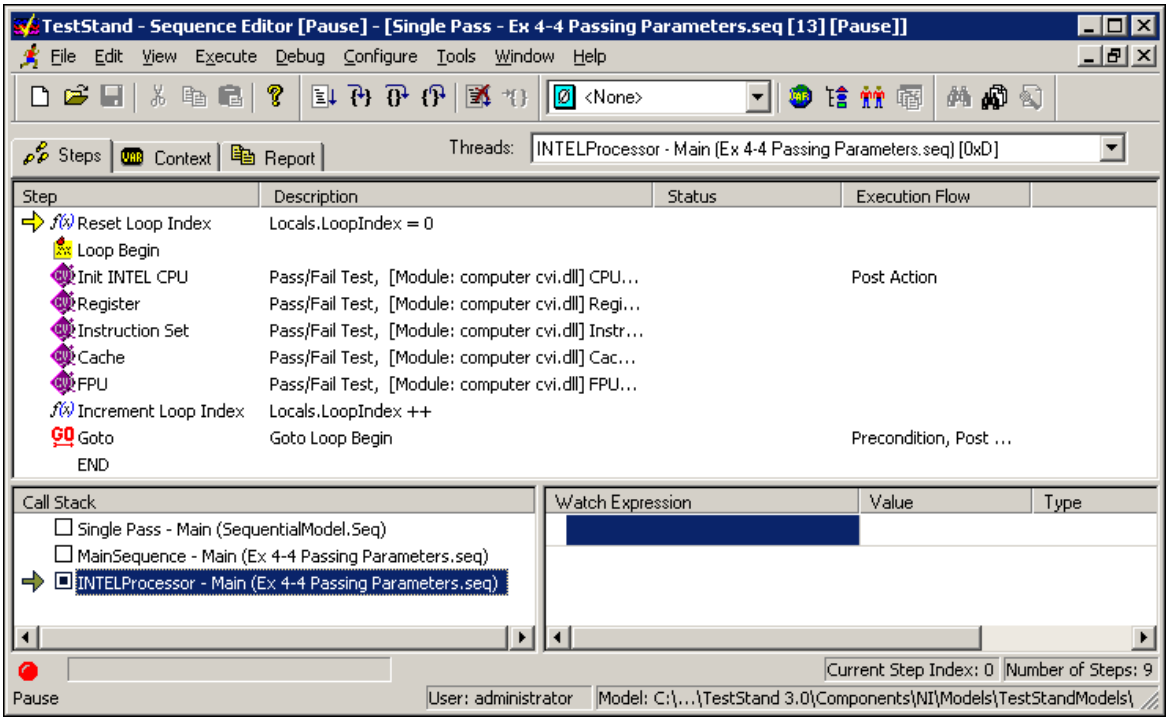


Figure 4-16. Call Stack Pane

The Call Stack pane shows the sequences that have been called. The most recently executed sequence appears at the bottom of the list.

7. Click the **Context** tab and select **Parameters**. Notice the values of the parameters for the sequence, as shown in Figure 4-17.

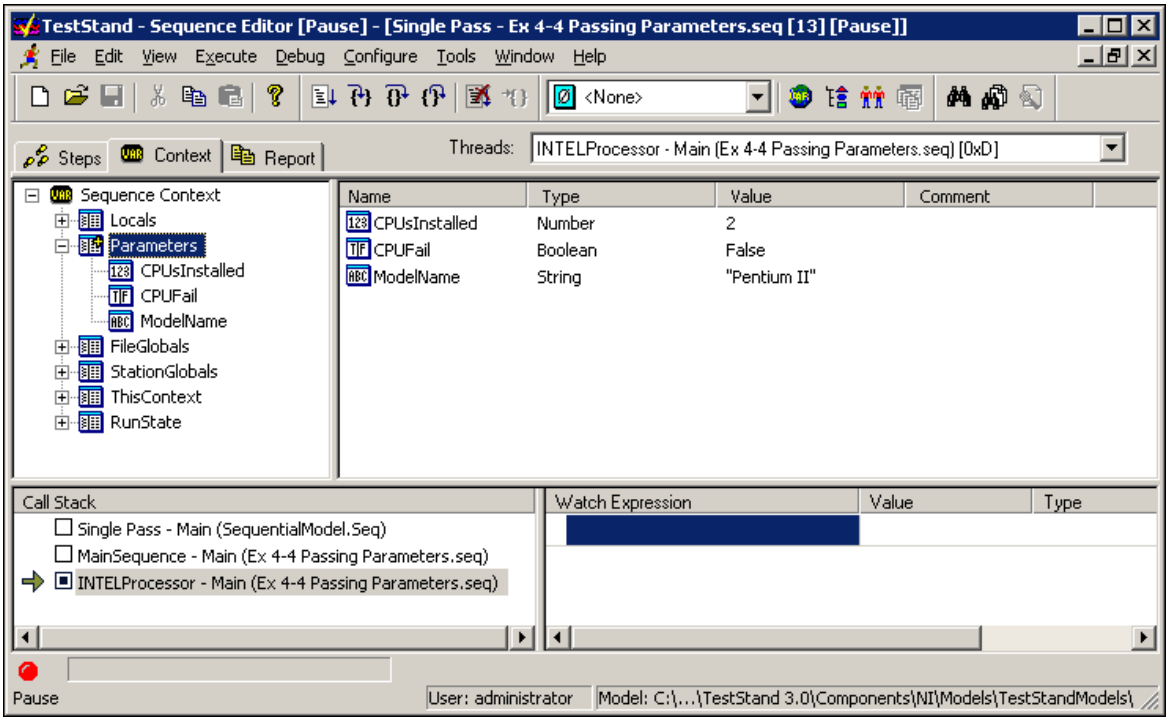


Figure 4-17. Context Tab of Sequence File

The value of the CPUsInstalled parameter is equal to the value you selected in the Specify Number of CPUs dialog box. Notice that MainSequence in the INTELProcessor.seq sequence file also requires a ModelName parameter. The Sequence Call step that you created did not specify this parameter, so the engine initializes the parameter value to its default value.

8. Select **Debug»Resume** and complete the execution.
9. When the execution completes, review the report to confirm that the correct sequence file was called, but do not close the Execution window.
10. Select **Execute»Restart** to restart the execution.
11. Click **Done** in the Test Simulator dialog box.
12. Click **AMD CPU** in the Select CPU dialog box.
13. Click **3** in the Number of CPUs dialog box.
14. When the execution pauses at the CPU Test step, single-step into the subsequence by selecting **Debug»Step Into**.

    Notice that the Call Stack pane lists AMDProcessor sequence at the bottom of the call stack.
15. Select **Debug»Resume to complete the execution**, and review the report.

16. Run the example a few more times and select other options. Each time, notice that the parameter values are passed to the dynamically called sequence file.
17. Close the Execution and Sequence File windows when you finish.

End of Part B

End of Exercise 4-4

Exercise 4-5 Dynamically Selecting Which Sequence to Run (Challenge)

Objective: To create a test that dynamically selects which sequence to run.

In this exercise, you will learn how to dynamically select a sequence file to run.

Open the `Ex 4-5 Dynamically Selecting Which Sequence to Run.seq` sequence file located in the `C:\Exercises\TestStand I` directory.

The Main sequence contains the following three steps—Generic Processor, Intel Processor, and AMD Processor. You should meet the following goals in this exercise.

- Add a step that asks the user to select the type of processor to test.
- Make the necessary changes to ensure that the selected processor is tested.

Hints:

- Use a Message Popup step to ask the user which type of processor to test.
- The Message Popup step stores the selected button in the following way: `RunState.Sequence.Main[`*`Name of the Message Popup step`*`].Result.ButtonHit`.
- Use the **Preconditions** expression of each step to determine which step will execute.

End of Exercise 4-5

Self Review

1. What are the four ways in which TestStand shares data?

2. Describe the difference in scope of the three types of TestStand variables.

3. When would you use a parameter instead of a variable?

4. Why are the values in a station global variable persistent?

5. What is contained with the RunState property object?

6. List and explain why some of the properties listed in the RunState property object are not available until run time.

7. What is the Expression Browser dialog box and how do you launch it?

8. All types of variables and parameters may be added using the Expression Browser dialog box. True or False

Notes

5

Creating Code Modules in External Environments

Lesson 5: Creating Code Modules in External Environments

In this lesson, you will:

- **Pass data between TestStand and external code modules**
- **Create code modules in LabVIEW and LabWindows/CVI**
- **Debug external LabVIEW or LabWindows/CVI code modules from TestStand**
- **Modify code templates using TestStand**

Introduction

This lesson describes how to create code modules in external environments for use in TestStand and how data is transferred from TestStand to other environments. This lesson also describes the LabVIEW and LabWindows/CVI test development environments, how to pass data to TestStand from these environments, and how to debug these code modules directly from TestStand.

Creating Code Modules in External Environments (Continued)

You can create tests in:

- **LabVIEW**
- **LabWindows/CVI**
- **Any environment capable of creating a DLL, EXE, ActiveX automation server, or subroutine**

Each type of code module uses a different adapter which acts to:

- **Execute code modules**
- **Step into code modules**
- **Generate code modules using templates**

Creating Code Modules in External Environments (Continued)

TestStand is a flexible testing environment that can call tests written in a wide variety of languages. TestStand can call any code module type for which it has an adapter. The adapter is an interface that allows the TestStand Engine to execute a code module and, in some cases, communicate with a specific programming language or file format. TestStand includes adapters for LabVIEW, LabWindows/CVI, DLLs, .NET Assemblies, ActiveX/COM automation servers, HTBasic subroutines, and TestStand sequences. Additionally, you can call Windows executable (`.exe`) files using a built-in step type.

Data Transfer in TestStand

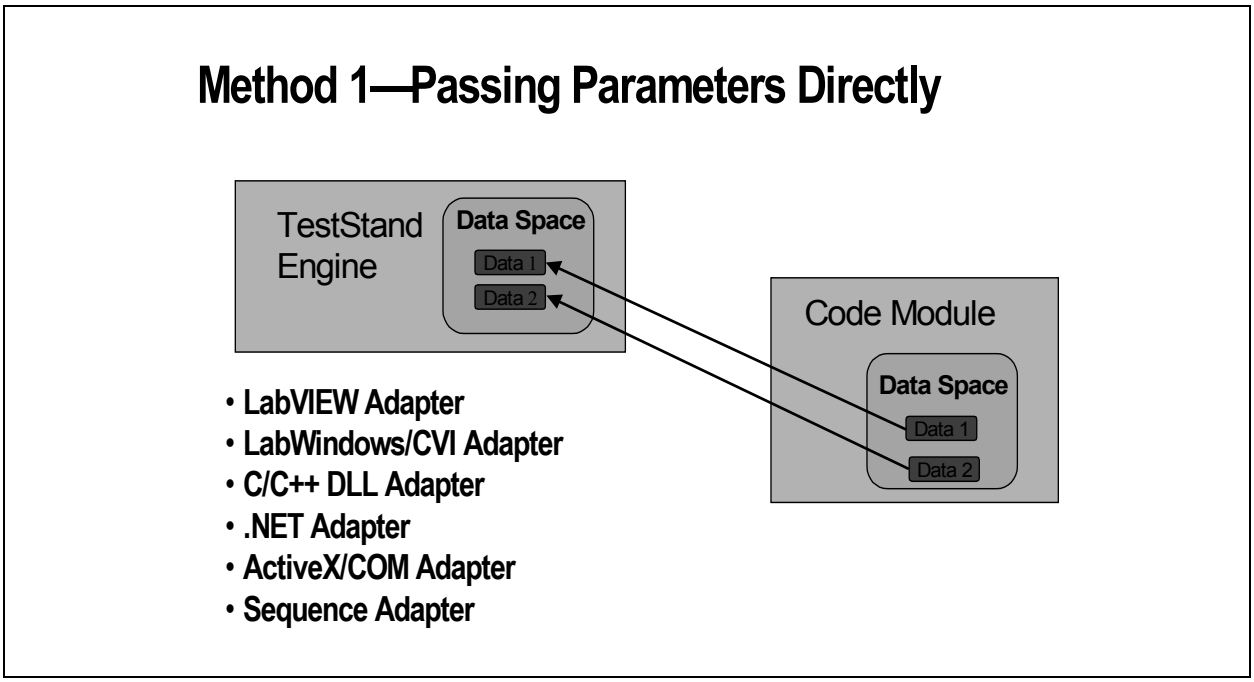
Pass data from an external code module to TestStand in one of the following ways:

- Pass an arbitrary number of parameters
- Use the TestStand ActiveX Automation API

Data Transfer in TestStand

There are two ways to pass data between an external code module and TestStand:

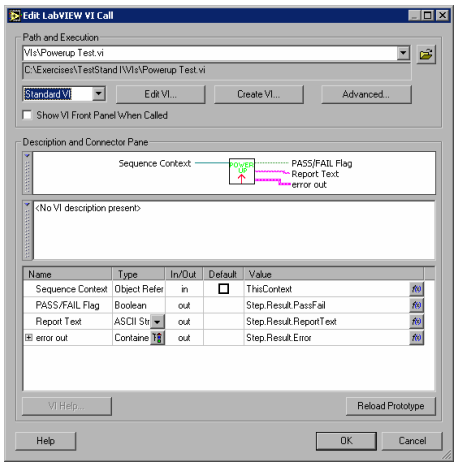
- Passing an arbitrary number of parameters using the LabVIEW Adapter, LabWindows/CVI Adapter, C/C++ DLL Adapter, .NET Adapter, ActiveX/COM Adapter, or Sequence Adapter.
- Using the TestStand ActiveX Automation API.



Method 1—Passing Parameters Directly

The LabVIEW, LabWindows/CVI, C/C++ DLL, .NET, ActiveX/COM, and Sequence Adapters can specify a type and arbitrary number of parameters that should be passed from the code module back to TestStand. Parameters pass the data among code modules and TestStand.

Passing Parameters - LabVIEW

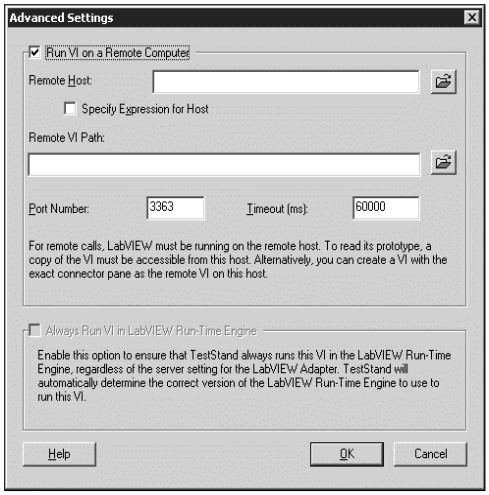


Passing Parameters – LabVIEW

TestStand allows users to call an arbitrary VI and pass TestStand data directly through the terminals on the VI connector pane. TestStand automatically loads the VI icon, connector pane information, and any VI help into the Specify Module dialog box so you can see exactly what parameters the LabVIEW VI expects and pass the appropriate values.

Note You must have LabVIEW 7.0 or later installed to use many of the TestStand 3.0 LabVIEW Adapter features.

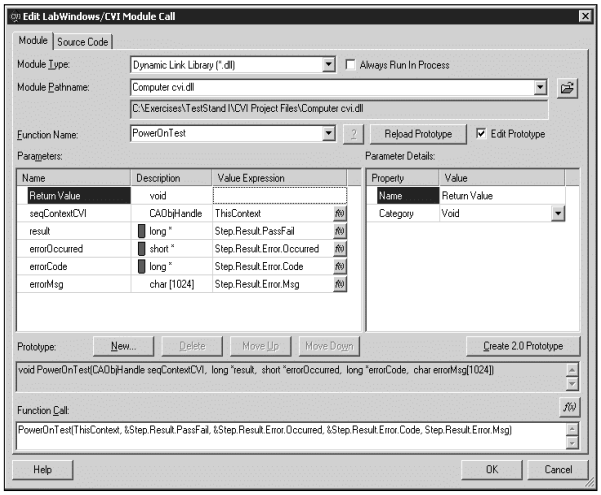
Passing Parameters – LabVIEW Advanced Settings



Passing Parameters – LabVIEW Advanced Settings

Click the **Advanced Settings** button on the Edit LabVIEW VI Call dialog box to configure TestStand for remote execution. Using a hostname, a user can specify a particular machine to execute the VI. The remote host can be another PC or a Real-Time Module target.

Passing Parameters – LabWindows/CVI



Passing Parameters – LabWindows/CVI

The LabWindows/CVI Adapter allows you to call different kinds of CVI modules, for example DLL, object, source, and static library, and provides the ability to step into CVI code directly from TestStand. Using the LabWindows/CVI Adapter, you can call functions with user defined function prototypes. The LabWindows/CVI Adapter allows you to pass a variety of data types from TestStand to LabWindows/CVI.

Exercise 5-1A or 5-1B

Exercise 5-1A: Creating Numeric Limit Tests with LabVIEW

Exercise 5-1B: Creating Numeric Limit Tests with LabWindows/CVI

Objective: To create a Numeric Limit Test using LabVIEW or LabWindows/CVI and pass the test results to a TestStand step.

Estimated Time: 15 minutes

Refer to page 5-26 for instructions for Exercise 5-1A.

Refer to page 5-34 for instructions for Exercise 5-1B.

Exercise 5-2A or 5-2B

Exercise 5-2A: Debugging Tests Using LabVIEW

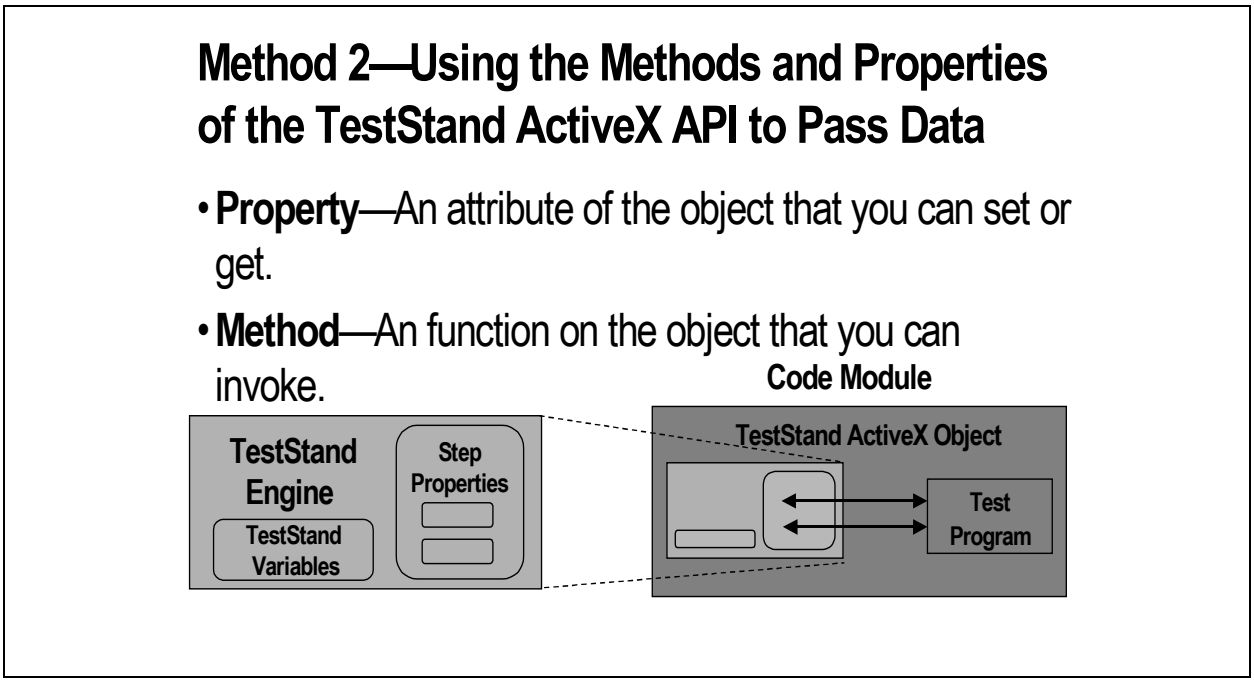
Exercise 5-2B: Debugging Tests Using LabWindows/CVI

Objective: To debug code modules compiled with LabVIEW or LabWindows/CVI.

Estimated Time: 10 minutes

Refer to page 5-41 for instructions for Exercise 5-2A.

Refer to page 5-44 for instructions for Exercise 5-2B.



Method 2—Using the Methods and Properties of the TestStand ActiveX API to Pass Data

You also can use ActiveX automation to pass data between a code module and TestStand. ActiveX automation is a standard method for passing data between Windows applications. It allows one application to pass data to or control another application by changing its properties and calling its methods. An ActiveX application can be thought of as an object used within another environment.

- **Property**—An attribute of the object that you can set or get.
- **Method**—A function of the object that you can invoke.

Using ActiveX, the SequenceContext object of a sequence can be passed to the code module. When executing sequences, TestStand maintains a sequence context that contains references to all global variables, local variables, and step properties in active sequences. The contents of the sequence context change depending on the currently executing sequence and step.

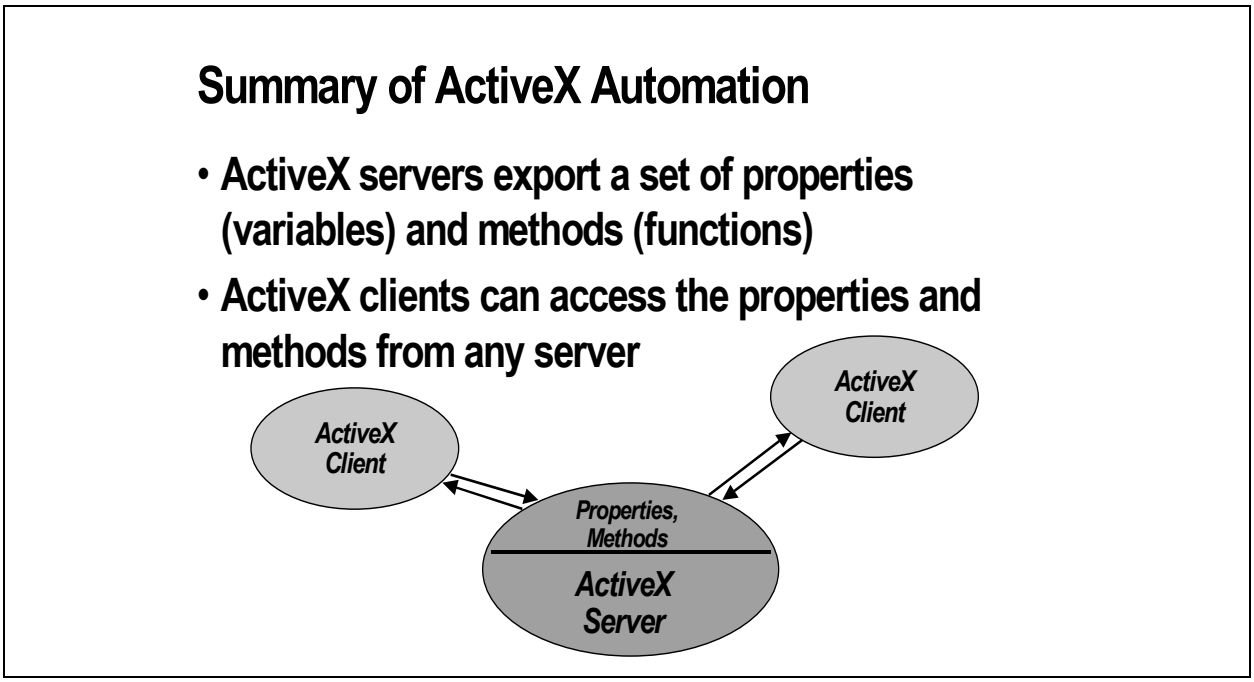
If you pass a SequenceContext object reference to the code module, you can use the TestStand ActiveX API to access the variables and properties in the sequence context. By setting properties and invoking methods of the SequenceContext object, you can pass data between the code module and the TestStand sequence, and you can set or change various TestStand attributes from within the code module.

What is ActiveX Automation?

- Microsoft technology for interapplication communication
- Allows one application to control or pass data to another application

What is ActiveX Automation?

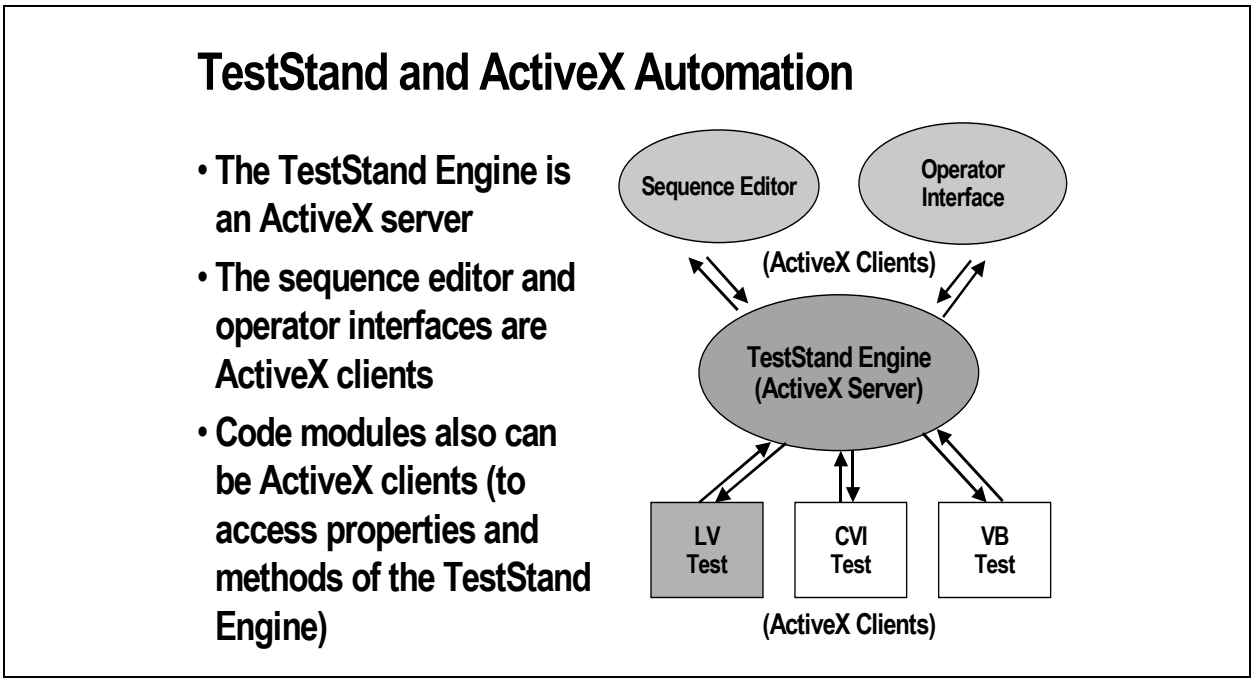
ActiveX is a set of Microsoft technologies widely used for interapplication communication. ActiveX gives you the freedom to share data between applications or programmatically control one application from another. It provides a standard interface and framework for communicating between different applications. Rather than developing a particular functionality within an application, you use ActiveX to leverage that functionality from another application.



Summary of ActiveX Automation

ActiveX applications can be either an ActiveX server or an ActiveX client. ActiveX servers export an Application Programming Interface (API) or a set of properties and methods to other applications. Think of properties as variables—their values can be read or set. Methods are like functions—they perform a certain action when invoked. ActiveX clients get or set properties and invoke methods of a server. Hence, ActiveX clients can control the behavior and update the variables of the ActiveX server.

Method 2 of passing data makes your code modules ActiveX clients to the TestStand ActiveX automation server. When you pass a SequenceContext object reference of the TestStand API to a TestStand code module, the TestStand code module can invoke properties and methods of the SequenceContext object to share data and control TestStand executions.



TestStand and ActiveX Automation

The TestStand Engine is an ActiveX server, which exports an ActiveX Application Programming Interface (API). The sequence editor and operator interfaces are ActiveX clients that use the engine to create, edit, execute, and debug sequences.

In addition, code modules also can access the TestStand API to get or set the values of TestStand variables or to invoke certain methods.

An example of a TestStand property is the value of a local or global variable. These values are maintained by the TestStand Engine, but can be accessed by an operator interface, sequence editor, or code module. DisplayLoginDialog and UnloadAllModules are examples of TestStand ActiveX API methods which ActiveX clients can call.

Lookup Strings

- Used to specify the dynamic property that a method operates on
- Defines the complete path from the object on which you call the method to the specific property you want to access
  Example: `Result.Error.Code`
- Can be copied from the Expression Browser dialog box or Context view of the executing sequence to the Windows clipboard

Lookup Strings

To specify the dynamic property that a method operates on, pass a string that defines a complete path from the object on which you call the method to the specific property you want to access. This string is called the lookup string. To specify the object itself, pass an empty string (`""`). To specify a subproperty, pass the name of the property. To specify the subproperty of a subproperty, pass a string containing both names separated by a period. For example, you can specify the error code property in a Step object using the following lookup string:

```
Result.Error.Code
```

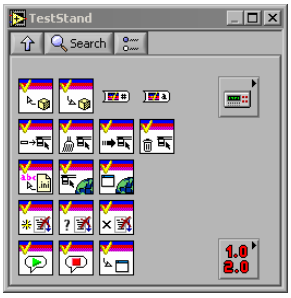
You can specify a complete path regardless of the length or complexity of the path, as long as all elements in the path are dynamic properties. As another example, to specify the error code in the third step of the Main step group of the currently executing sequence, you can use the SequenceContext object with the following lookup string:

```
RunState.Sequence.Main[2].Result.Error.Code
```

You can copy lookup strings from the Expression Browser dialog box or the Context view of an executing sequence. You can paste this string into your code module, or wherever you want to use it.

ActiveX Automation in LabVIEW

- TestStand installs a palette of ActiveX VIs
- These VIs access common TestStand properties
- For all other properties and methods use the LabVIEW Property Nodes and Invoke Nodes

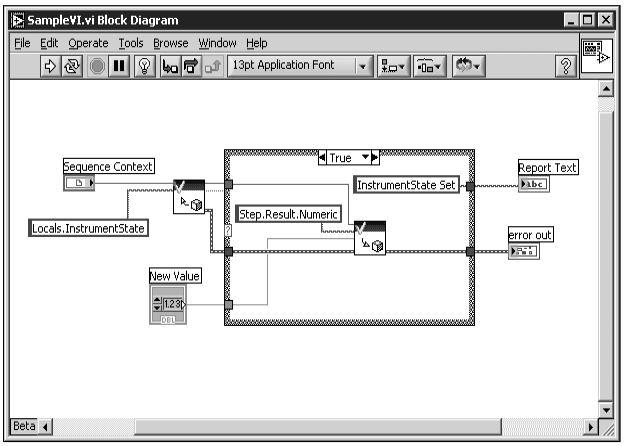


ActiveX Automation in LabVIEW

TestStand installs a palette of VIs for accessing common TestStand properties, including getting or setting the value of any TestStand numeric, string, Boolean, or array. You can access all other TestStand properties and methods using the Invoke Node and Property Node located on the **Communication»ActiveX** palette. In fact, if you open the VIs provided in the TestStand palette, you will find the LabVIEW Invoke Nodes and Property Nodes. These nodes illustrate the general manner of accessing TestStand properties. You can find additional information about the LabVIEW ActiveX capability in the *LabVIEW User Manual*.

Note To use TestStand 3.0 VIs with LabVIEW, you must have LabVIEW 6.1 or later.

Example: ActiveX Automation in LabVIEW



Example: ActiveX Automation in LabVIEW

The slide above shows an example of using ActiveX automation within LabVIEW to get and set the value of TestStand variables. The ActiveX Automation handle, SequenceContext, is a refnum passed to the VI from TestStand. It gives the VI a reference to the TestStand sequence that called it. The first subVI, Get Property Value, gets the value of a TestStand local variable, in this case the Instrument State Boolean variable. The second subVI, Set Property Value, sets the value of a subproperty of the step that called this code module. The subproperty in this case is a numeric value named Numeric.

ActiveX Automation in LabWindows/CVI

TestStand API instrument driver

- **TestStand provides a LabWindows/CVI instrument driver to access TestStand properties and methods**
- **The instrument driver is a LabWindows/CVI function panel file**
- **TestStand properties and methods are set by calling the appropriate functions from within LabWindows/CVI**

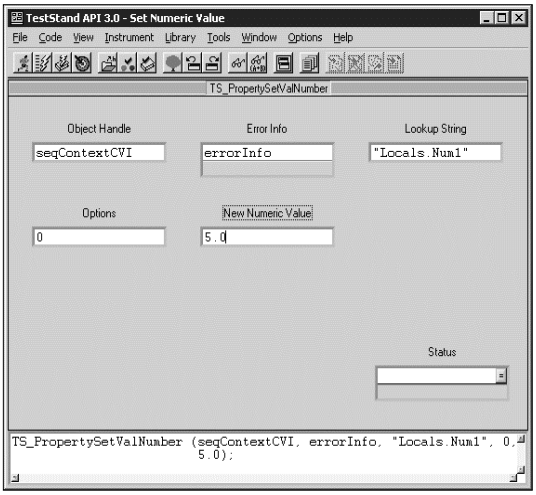
ActiveX Automation in LabWindows/CVI

The instrument driver, `<TestStand>\API\CVI\tsapicvi.fp`, is a set of function panels and modules containing functions for accessing all TestStand properties and methods. Remember that when you are in the LabWindows/CVI environment, you might need functions from the LabWindows/CVI ActiveX automation library to convert data types returned from the TestStand instrument driver.

It is also important to remember that by using ActiveX automation, you have access to not only the data relevant to the specific code contained within a test, but also to many of the TestStand environment attributes and functions, thereby allowing you to make changes to sequence information during run time, as well as giving you control over TestStand executions.

TestStand also provides a utility instrument driver, `<TestStand>\API\CVI\tsutil.fp`, that supports a set of functions that further simplify TestStand programming in LabWindows/CVI. The utility instrument driver combines the basic TestStand instrument driver functionality with the existing LabWindows/CVI functionality including user interface, toolbox, and error handling.

Example: ActiveX Automation in LabWindows/CVI



Example: ActiveX Automation in LabWindows/CVI

This slide above shows an example of a LabWindows/CVI function panel for the PropertySetValNumber function. It takes in a LabWindows/CVI sequence context passed by the TestStand step that calls the code module, a lookup string, and a new numeric value. This example sets the value of the local variable Num1 to `5.0`.

In Exercise 5-3, you will use ActiveX to pass data between TestStand and your code module.

Exercise 5-3A or 5-3B

Exercise 5-3A: Using the ActiveX API in LabVIEW Code Modules

Exercise 5-3B: Using the ActiveX API in LabWindows/CVI Code Modules

Objective: To use the ActiveX API to pass data from your code module to TestStand.

Estimated Time: 20 minutes

Refer to page 5-47 for instructions for Exercise 5-3A.

Refer to page 5-52 for instructions for Exercise 5-3B.

Differences in the C-Language Adapters

- Both the C/C++ DLL Adapter and the LabWindows/CVI Adapter call DLLs
- Similarities and differences between the C/C++ DLL Adapter and the LabWindows/CVI Adapter

Differences in the C-Language Adapters

Both the LabWindows/CVI Adapter and the C/C++ DLL Adapter can call C-language DLLs, but they use different methods to call these code modules. This discussion compares these two adapter types and describes the benefits of each.

The LabWindows/CVI Adapter allows you to step into C source files and DLLs from TestStand using an external instance of LabWindows/CVI. The LabWindows/CVI Adapter makes debugging more convenient for LabWindows/CVI users by allowing them to step into their C source or DLL directly from TestStand.

If you are using Microsoft Visual Studio 7.0 or higher and debugging in TestStand, the **Step Into** debugging command launches Microsoft Visual Studio. Execution then breaks on the first statement in the called function or method.

You can always debug a DLL from your C/C++ development environment by launching the sequence editor or operator interface as an external process from the development environment.

Differences in the C-Language Adapters (Continued)

Feature	LabWindows/ CVI Adapter	C/C++ DLL Adapter
Test functions can have a variety of prototypes	Yes	Yes
Allows you to pass any argument to a test function directly	Yes	Yes
Automatically passes a predefined set of common arguments to each test function	No	No
Code modules can be C source files, object files, or static libraries	Yes	No (DLLs only)
Can call DLLs created with MSVC++, Borland C++, Delphi, or another compiler	No	Yes
Backward compatible with LabWindows/CVI Test Executive Toolkit	Yes	No

Continued on next slide

Differences in the C-Language Adapters (Continued)

The LabWindows/CVI Adapter also allows you to use object files, static libraries, and C source files in LabWindows/CVI. In addition, the LabWindows/CVI Adapter has tighter integration with the LabWindows/CVI environment for creating and editing code.

When using the C/C++ DLL adapter or older versions of TestStand and LabWindows/CVI, you must provide a type library if you want TestStand to populate parameter and protytpe information. If you are using LabWindows/CVI 7.1 or later and TestStand 3.1 or later, function information will be embedded within the DLL itself and you do not need to create a type library for the parameters and prototypes to appear.

Keep in mind that your test sequence can contain a mixture of both adapters.

Differences in the C-Language Adapters (Continued)

Feature	LabWindows/ CVI Adapter	C/C++ DLL Adapter
Allows debugging C source files from TestStand	Yes [1,2]	No
Allows debugging DLL functions from TestStand	Yes [1,2]	Yes [1,3]
Can extract function information from DLL	Yes [5]	No
Can create source code from code templates	Yes [4]	Yes

1. Requires LabWindows/CVI 6.0 or later.
2. You must configure the LabWindows/CVI Adapter to execute steps in an external instance of LabWindows/CVI.
3. You must launch the sequence editor or operator interface from the DLL development environment.
4. Creation and editing of source code is integrated with the LabWindows/CVI environment.
5. Requires LabWindows/CVI 7.1 or later and TestStand 3.1 or later.

Differences in the C-Language Adapters (Continued)

When using the C/C++ DLL adapter or older versions of TestStand and LabWindows/CVI, you must provide a type library if you want TestStand to populate parameter and prototype information. If you are using LabWindows/CVI 7.1 or later and TestStand 3.1 or later, function information will be embedded within the DLL itself and you need not create a type library for the parameters and prototypes to appear.

Using TestStand Code Templates

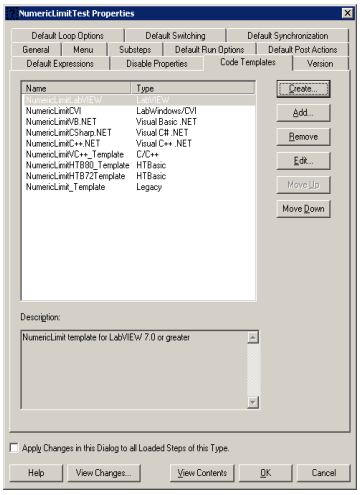
- **TestStand allows test developers to create code in an external environment from within the sequence editor**
- **Each adapter and step type has an associated code template**
  - Use the code template to create default test code
  - Use the Create Code feature to eliminate redundant programming

Using TestStand Code Templates

Another powerful feature of TestStand is the ability to create tests in LabVIEW or LabWindows/CVI from within the TestStand Sequence Editor. Select the **Create Code** option in the Specify Module dialog box to create code using a default code template for the respective programming environment. TestStand allows you to customize this template to include any necessary code for each test of a certain type, or to associate a number of templates with each step type. If you define more than one template for a step type, TestStand prompts you to choose one from a list when you select Create Code. This feature allows you to develop your test sequences in an efficient and consistent way.

Code Templates

- Code templates are configured through the step type properties found in the Type Palette
- Code for the code templates is stored in the `<TestStand>\CodeTemplates` directory



Code Templates

Create and configure code templates on the **Code Templates** tab of the Step Properties dialog box. The actual code for each template is in the `<TestStand>\CodeTemplates` directory.

Lesson 5 Summary: Creating Code Modules in External Environments

There are two ways to pass data between the TestStand Engine and code modules:

- **Passing arbitrary number of parameters**
- **Using the ActiveX Automation API exported by the TestStand Engine**

TestStand eliminates redundant programming by allowing you to specify default code based on code templates

Summary

This lesson described how to share data between TestStand and code modules developed in another programming environment. You can use an arbitrary number of parameters, or you can leverage the ActiveX capabilities of the engine (server) and development environments such as LabVIEW, LabWindows/CVI, or Microsoft Visual Basic (clients).

TestStand also includes a method to reduce redundant programming. Using code templates, you can duplicate common components used throughout your code modules.

Note If you are using either LabWindows/CVI or LabVIEW, you can use National Instruments IVI driver technology to perform test software development without requiring instrumentation hardware. This instrument driver technology provides features such as state caching, simulation, and an attribute model for convenient test development. Refer to Appendix A, *Introduction to IVI*, for more information about IVI.

Exercise 5-1A Creating Numeric Limit Tests with LabVIEW

Objective: To create a Numeric Limit Test using LabVIEW and pass the test results to a TestStand step.

This exercise shows how to create a Numeric Limit Test in TestStand and demonstrates one way to add information to the test report. The code module passes numeric data and a Report Text string back to TestStand. TestStand compares the numeric data to predefined limits and determines whether the test passed or failed. The Report Text string displays in the report.

Part A: Creating the LabVIEW Test

This exercise describes how to create a LabVIEW test that returns a random numeric value and a Report Text string. The default limits of `0` for minimum and `10` for maximum are used to evaluate the pass/fail conditions based on the numeric measurement returned from the test.

1. In the sequence editor, select **File»Open** and open the `Ex 5-1A Creating Numeric Limit Tests with LabVIEW.seq` file located in the `C:\Exercises\TestStand I` directory.

   This sequence file is similar to ones used in previous exercises, except it is missing the Video Diagnostics step.

2. Select the **LabVIEW Adapter** from the **Adapter** ring control.

3. Right-click the `RAM Diagnostics` step and select **Insert Step»Tests» Numeric Limit Test** from the context menu.

   This inserts a Numeric Limit Test step for the LabVIEW Adapter between the RAM Diagnostics step and the Keyboard Diagnostics step.

4. Name this new step `Video Diagnostics`.

5. To view the limits, right-click the `Video Diagnostics` step and select **Edit Limits** from the context menu. Figure 5-1 shows the Edit Numeric Limit Test dialog box.

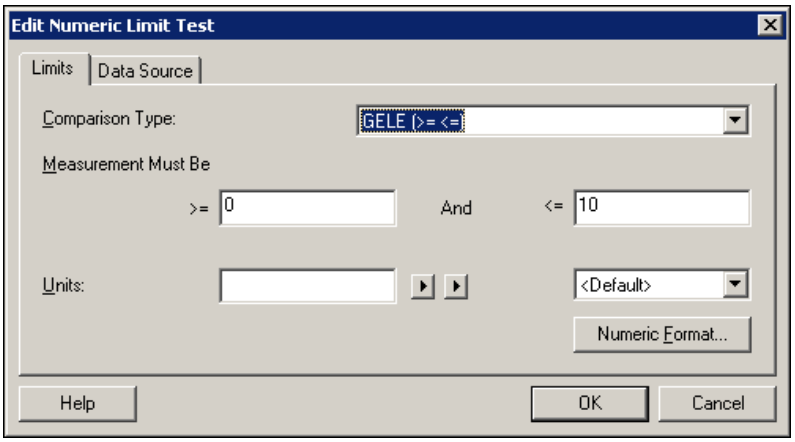


Figure 5-1. Edit Numeric Limit Test Dialog Box

6. Click **OK** to return to the Sequence File window.
7. Right-click the `Video Diagnostics` step and select **Specify Module** from the context menu to open the Edit LabVIEW VI Call dialog box.
8. Enter `VIs\VideoDiagnostics.vi` in the **VI Pathname** control of the Edit LabVIEW VI Call dialog box, as shown in Figure 5-2.

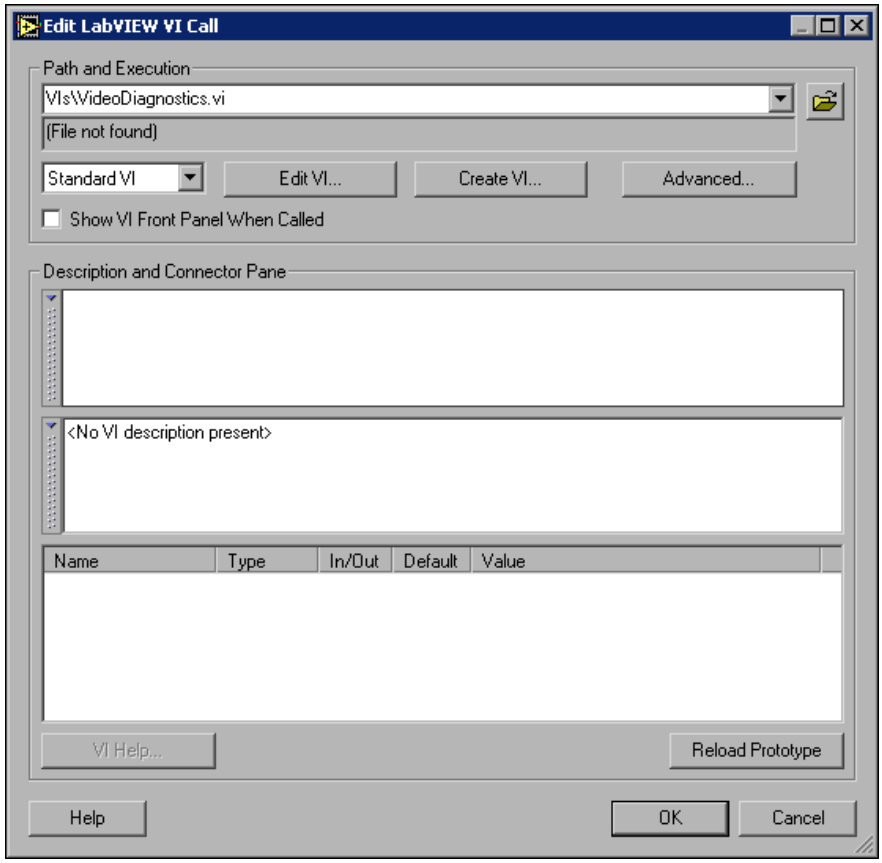


Figure 5-2. Edit LabVIEW VI Call Dialog Box

9. Click the **Create VI** button and create a code for this VI
10. LabVIEW launches `VideoDiagnostics.vi` and displays the front panel, as shown in Figure 5-3.

    Notice that the VI already contains the **Numeric Measurement** indicator, **Report Text** indicator, and **error out** cluster. These objects contain information that the step passes to TestStand.

    Because you used the **Create VI** button to create the VI, TestStand automatically builds the VI with the required components for this kind of step, in this case a Numeric Limit Test.

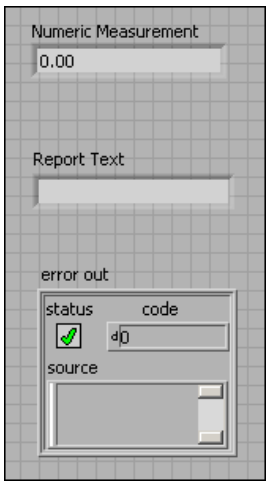


Figure 5-3. Video Diagnostics VI

11. Create inputs to the VI. Right-click the front panel to display the Controls palette and select **Numeric Controls»Numeric Control** to place a numeric control on the front panel.
12. Name the control `Input1`.
13. Repeat steps 11 and 12 to create a `MaxInput` control. The front panel should look similar to Figure 5-4.

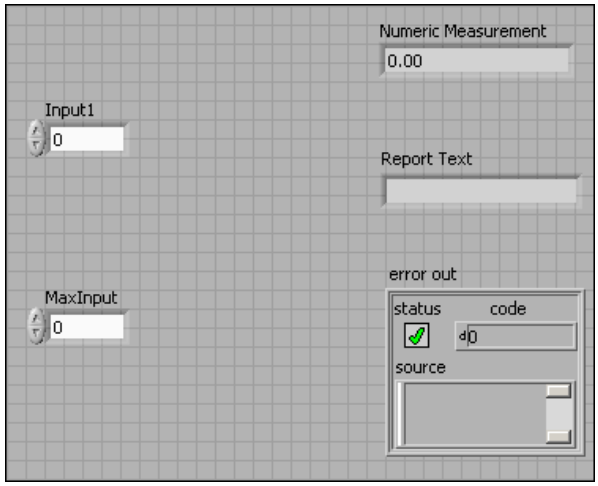


Figure 5-4. Video Diagnostics VI with Input Controls

14. Select **Window»Show Block Diagram** to open the block diagram and build the diagram as shown in Figures 5-5 and 5-6.

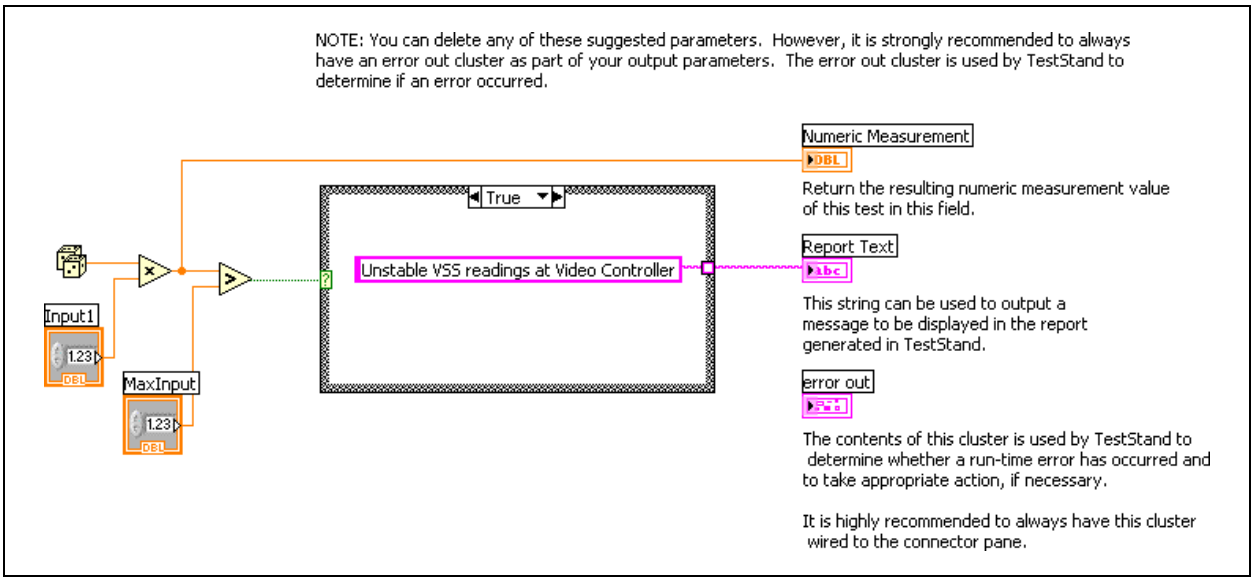


Figure 5-5. True Case Structure

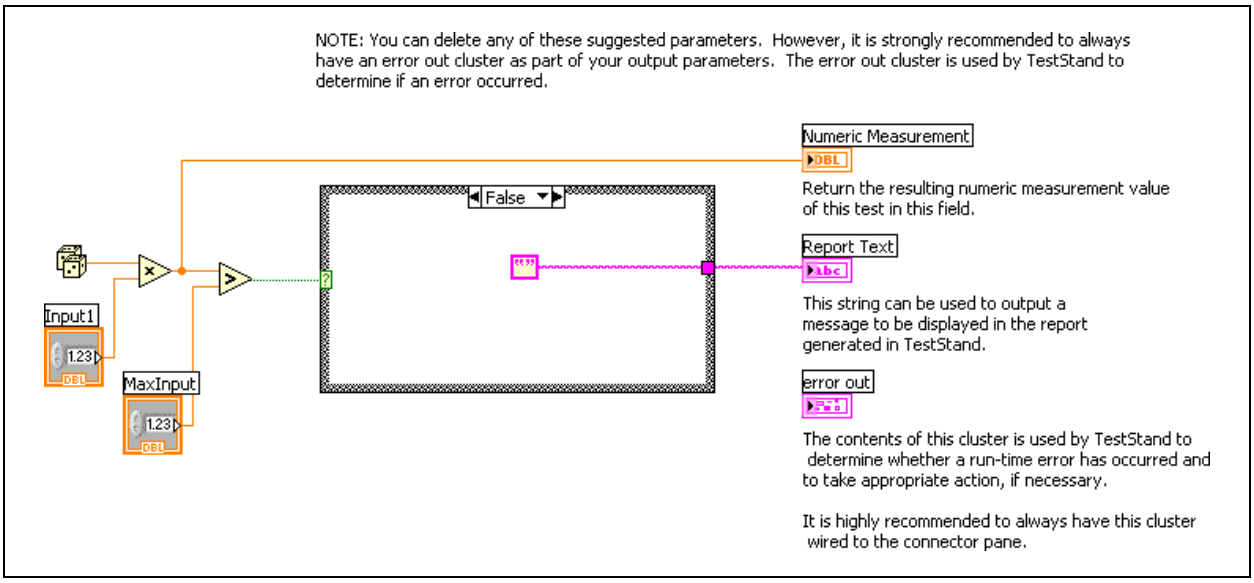


Figure 5-6. False Case Structure

Additional Information

This VI simulates a test that generates a random number between the values of `0` and `Input1` by multiplying the output of the Random Number (0-1) function by the `Input1` value. The random number then passes into the Numeric Measurement terminal. This value is compared to the value of `MaxInput`; if the value is greater than `MaxInput`, a string is sent to the Report Text terminal.

15. Select **Window»Show Front Panel** to display the front panel.
16. Right-click the VI icon in the upper right corner of the front panel and select **Show Connector** from the context menu to display the connector pane for the VI.
17. Click the upper left terminal of the connector pane, then click the `Input1` numeric control to assign this terminal to the `Input1` control. The connector pane terminal should turn orange to represent a numeric input.
18. Click on the next connector pane terminal below the orange one to select it, then click the `MaxInput` control to assign this terminal to the `MaxInput` control. The connector pane terminal should turn orange to represent a numeric input.
19. Select **File»Save As** and save the completed VI as `VideoDiagnostics.vi` in the `C:\Exercises\TestStand I\VIs` directory.
20. Close the VI and return to the sequence editor.
21. In the Edit LabVIEW VI Call dialog box, click **Reload Prototype**. When TestStand prompts you to retain current parameter values, click the **Yes** button. The controls you added to the VI are visible.

22. Configure the Edit LabVIEW VI Call dialog box as shown in Figure 5-7.

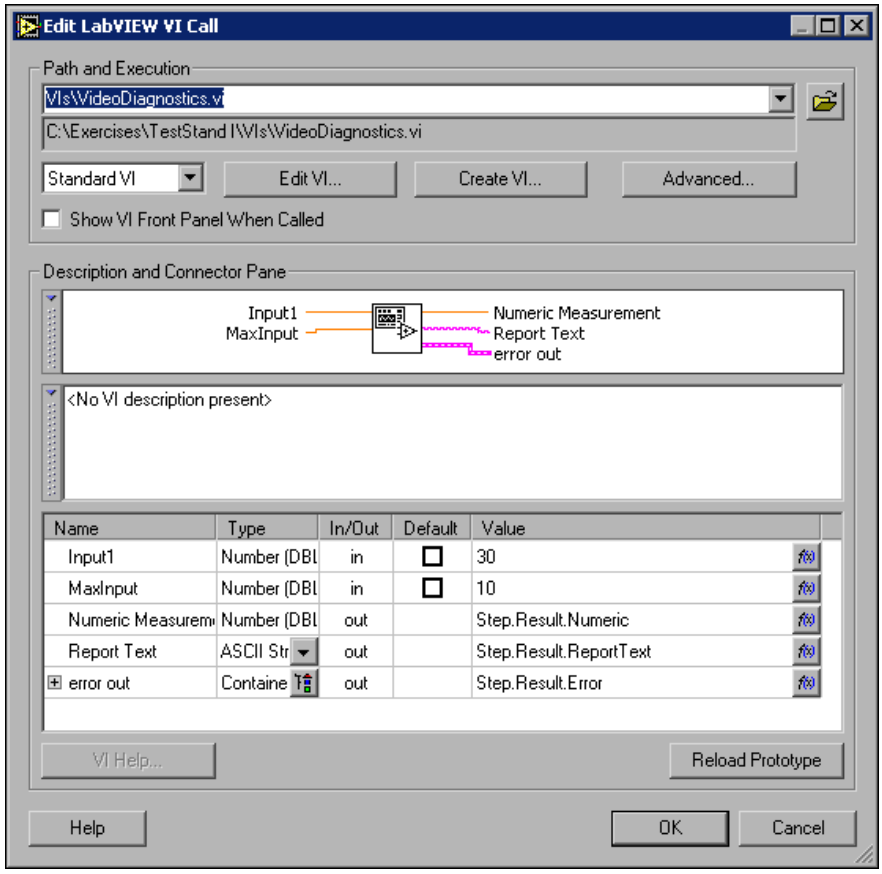


Figure 5-7. Completed Edit LabVIEW VI Call Dialog Box

Notice that you could pass a constant or a variable value to the LabVIEW VI. In this case, you use constants.

23. Click **OK** to close the Edit LabVIEW VI Call dialog box.

End of Part A

Part B: Configuring the Step Properties

Now that you have created the step and added the code, configure the preconditions for this step. The Video Diagnostics step should execute only if the Video Test step fails and the Powerup Test and CPU Test pass.

1. Double-click the `Video Diagnostics` step to open the Step Properties dialog box.
2. Click **Preconditions** and set the preconditions.
   a. Select the **Powerup Test** in the **Insert Step Status** section.
   b. Click **Insert Step Pass**.

c. Select the **CPU Test** and click **Insert Step Pass**.

d. When prompted to **InsertAllOf** or **InsertAnyOf**, select **InsertAllOf**.

Note Selecting **InsertAllOf** instructs TestStand that all of the the preconditions must be true in order to execute the step. If you select **InsertAnyOf**, the step executes if any of the preconditions are true.

e. Select the **Video Test** and click **Insert Step Fail**.

Figure 5-8 shows the completed Preconditions dialog box.

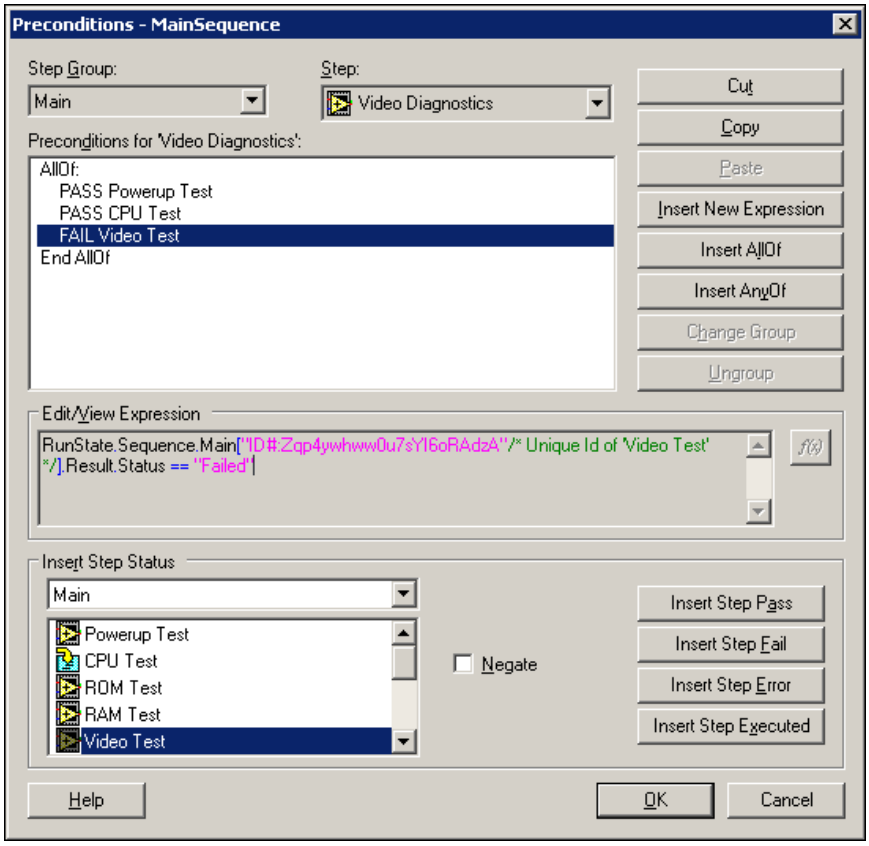


Figure 5-8. Precondition Settings

3. Click **OK** to close the Preconditions dialog box.
4. Click **OK** to close the Step Properties dialog box for the Video Diagnostics step.
5. Save the sequence.
6. Execute the sequence by selecting **Execute»Single Pass**.
7. When prompted during execution, select the Video test to fail.

8. When the report generates, scroll down the report and observe that the Video test is reported as having failed. Continue scrolling down and notice the status of the Video Diagnostics step. The measurement value returned should be between 0 and 30. Observe the Report Text string returned from the code module if the value is greater than 10.

End of Part B

End of Exercise 5-1A

Exercise 5-1B Creating Numeric Limit Tests with LabWindows/CVI

Objective: To create a Numeric Limit Test using LabWindows/CVI and pass the test results to a TestStand step using structures.

This exercise shows how to create a Numeric Limit Test in TestStand and demonstrates one way to add information to the Test Report. The code module passes numeric data and a Report Text string back to TestStand. TestStand compares the numeric data to predefined limits and determines whether the test passed or failed. The Report Text string displays in the report.

Part A: Creating the LabWindows/CVI Test

In the following steps you will create a DLL code module that returns a random numeric measurement and a Report Text string. Use the default limits of `0` for minimum and `10` for maximum to evaluate the pass/fail conditions based on the numeric measurement returned from the test.

1. Select **File»Open** from the sequence editor and open the `Ex 5-1B Creating Numeric Limit Tests with CVI.seq` file located in the `C:\Exercises\TestStand I` directory.

   This sequence file is similar to those used in previous exercises, except it is missing the Video Diagnostics step.

2. Select the **LabWindows/CVI Adapter** from the **Adapter** ring control.
3. Right-click the `RAM Diagnostics` step and select **Insert Step»Tests» Numeric Limit Test** from the context menu to insert a Numeric Limit Test step using the LabWindows/CVI Adapter between the RAM Diagnostics step and the Keyboard Diagnostics step.
4. Name this step `Video Diagnostics`.
5. To view the limits, right-click the test and select **Edit Limits** from the context menu. Observe the current limits.
6. Click **OK** to return to the Sequence File window.
7. Right-click the `Video Diagnostics` step and select **Specify Module** from the context menu.
8. Configure the Edit LabWindows/CVI Module Call dialog box as shown in Figure 5-1.

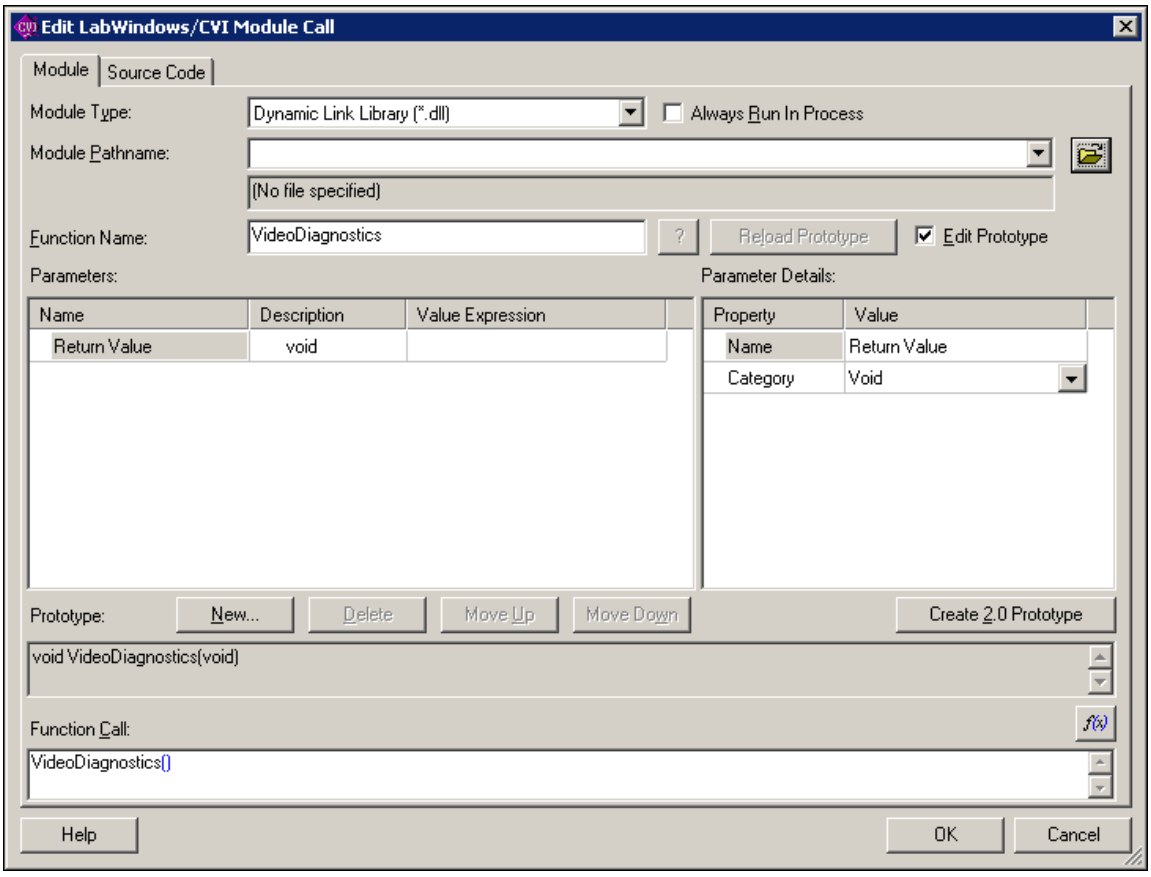


Figure 5-1. Module Tab Settings

9. Click the **Source Code** tab and complete the following settings.
   a. Enter `CVI Project Files\Video Diagnostics.c` in the **Source File Containing Function** field.
   b. Enter `CVI Project Files\Video Diagnostics.prj` in the **CVI Project File to Open** field.

   Notice that the source code file and the project file do not yet exist. The remaining steps in this exercise show you how to create these files. The **Source Code** tab of the Edit LabWindows/CVI Module Call dialog box should now appear as shown in Figure 5-2.

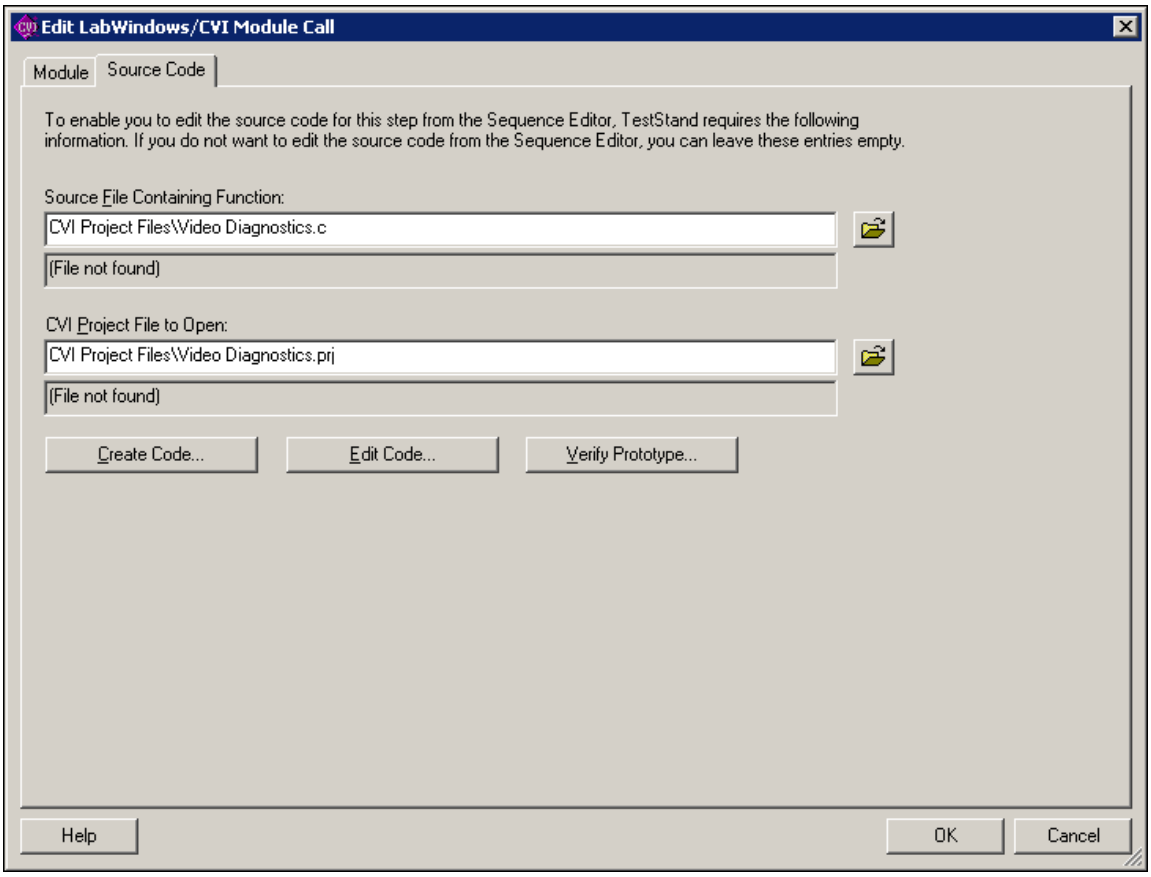


Figure 5-2. Source Code Tab Settings

10. Click **Create Code**.
11. When prompted, click **OK** to confirm that the project file and source file should be created in the `C:\Exercises\TestStand I\CVI Project Files` subdirectory. Make sure this subdirectory is selected; if it is not selected, select this directory before clicking **OK**.
12. If prompted, select the **Use a Relative Path** option and click **OK** to continue.
13. LabWindows/CVI launches and creates the `Video Diagnostics.c` source file, shown in Figure 5-3. This source file contains the template for the function that you must create, called VideoDiagnostics.

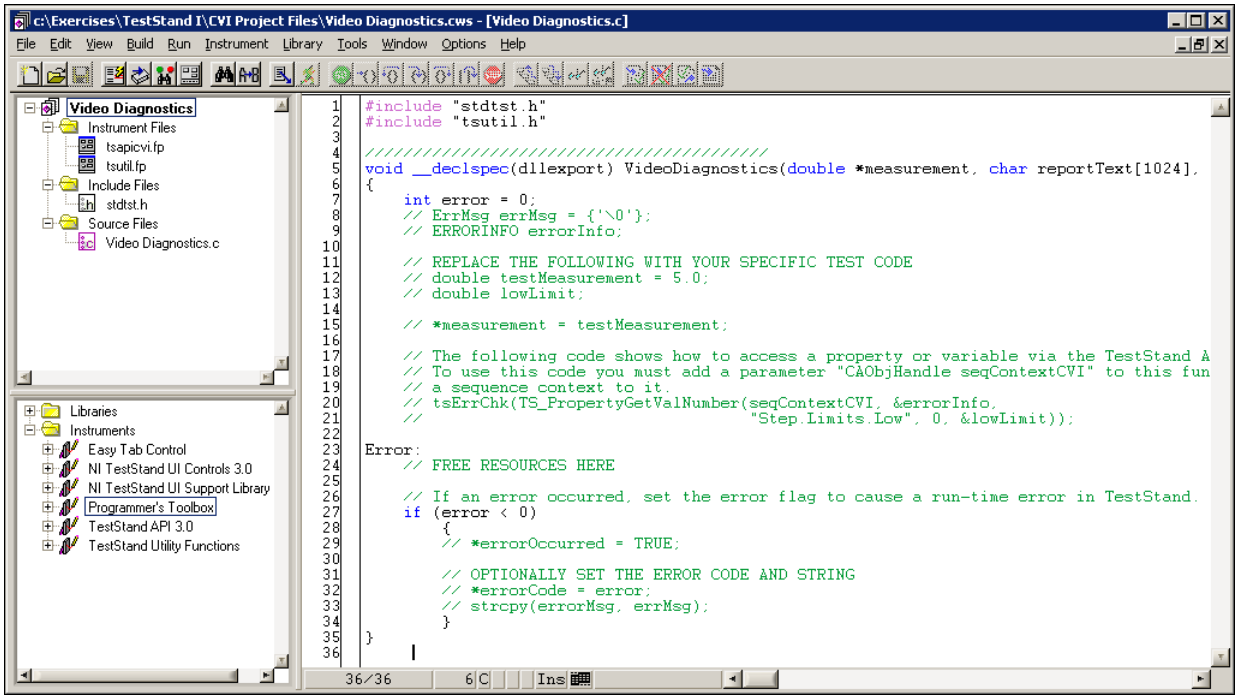


Figure 5-3. Video Diagnostics Source File

14. The comments are placeholders for you to insert your code. Modify this function to generate a random number and store it in the measurement parameter. Make sure the code appears exactly as shown in Figure 5-4.

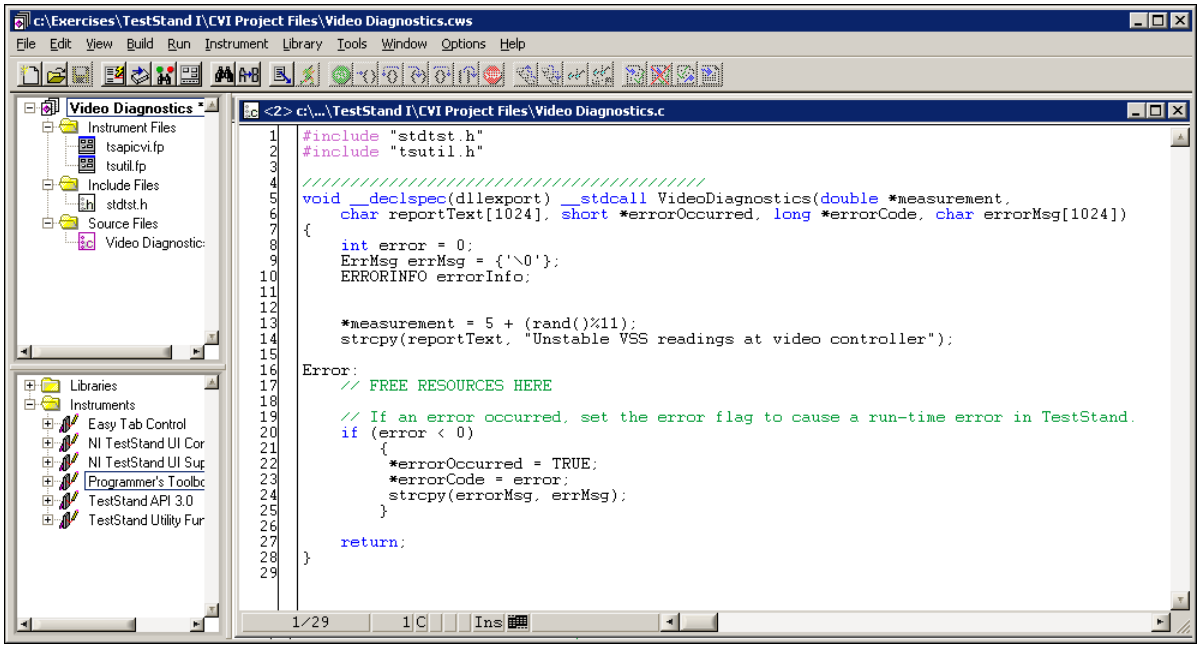


Figure 5-4. Modified Source File

15. Save the source file as `VideoDiagnostics.c` in the `C:\Exercises\TestStand I\CVI Project Files` directory.
16. Set the project target to **Dynamic Link Library** under the **Build» Target Type** menu.
17. Select **Build»Target Settings** to launch the Target Settings dialog box and confirm that the DLL target setting is correct. Verify the settings against those shown in Figure 5-5.

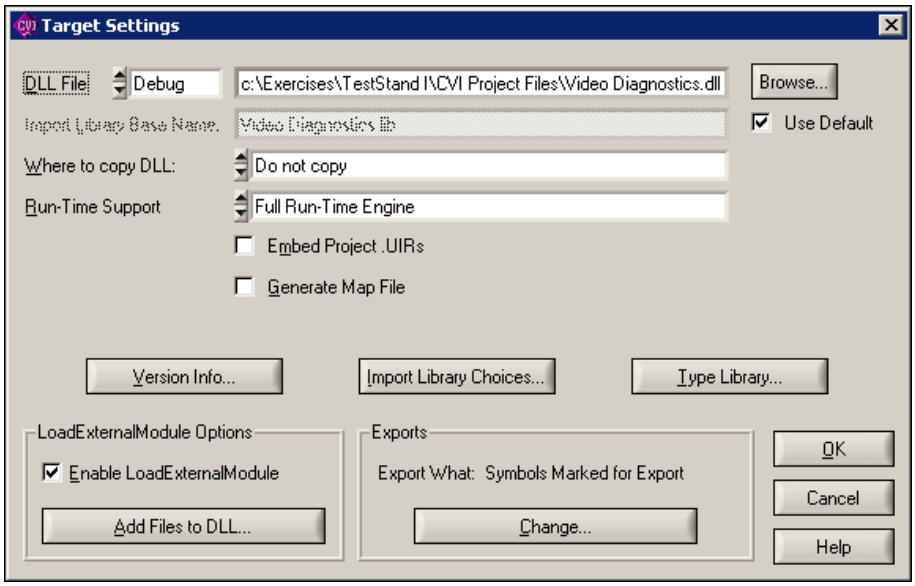


Figure 5-5. Target Settings Dialog Box

18. Click **OK** twice to close the DLL Export Options and Target Settings dialog boxes.
19. Select **Build»Create Debuggable Dynamic Link Library** to create the DLL.

    LabWindows/CVI creates three files during the build—`Video Diagnostics.dll`, `Video Diagnostics.cdb`, and `Video Diagnostics.lib`. Click **OK** to continue.
20. Return to the Edit LabWindows/CVI Module Call dialog box in TestStand.

    Notice that the modules are located in the proper directories in the pathname text boxes, because they have been created.
21. Click **OK** to exit the Edit LabWindows/CVI Module Call dialog box.

End of Part A

Part B: Configuring the Step Properties

Add the precondition to the step. The Video Diagnostics step should execute only if the Video Test step fails and the Powerup Test and CPU Test steps pass.

1. Double-click the `Video Diagnostics` step to open the Step Properties dialog box.
2. Click **Preconditions** to open the Preconditions dialog box and set the following preconditions:
   a. Select the **Powerup Test** in the **Insert Step Status** section.
   b. Click **Insert Step Pass**.
   c. Select the **CPU Test** and click the **Insert Step Pass** button.
   d. When prompted to **InsertAllOf** or **InsertAnyOf**, select **Insert AllOf**.

Note Selecting **InsertAllOf** instructs TestStand that all of the the preconditions must be true in order to execute the step. If you instead select **InsertAnyOf**, the step executes if any of the preconditions are true.

   e. Select the **Video Test** and click **Insert Step Fail**.

Figure 5-6 shows the completed Preconditions dialog box.

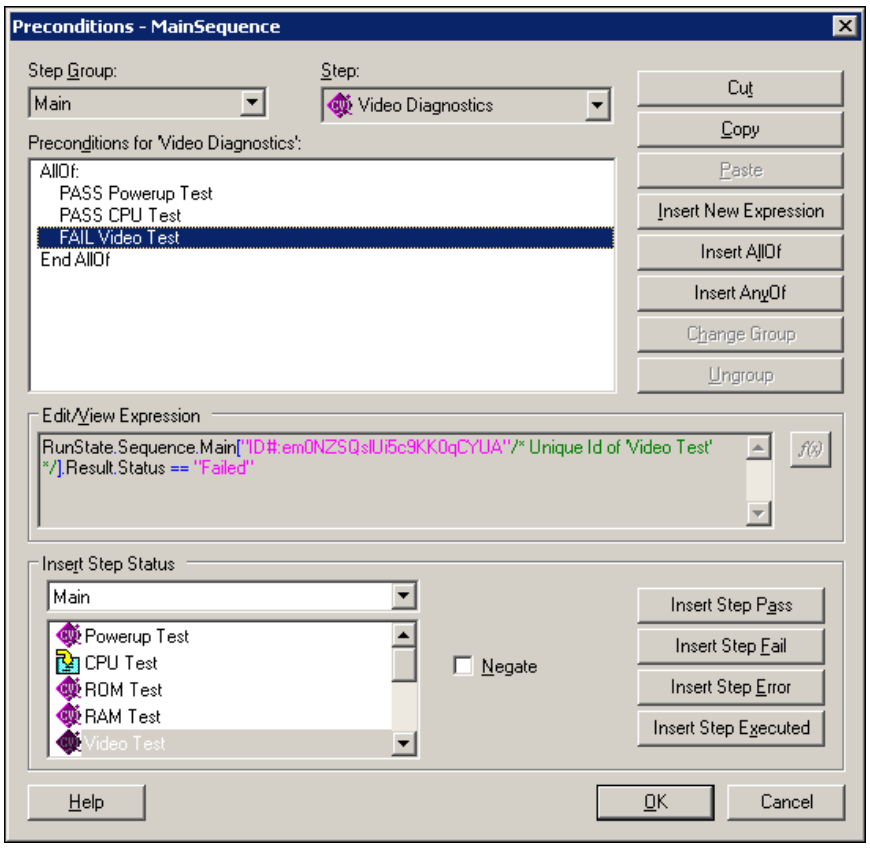


Figure 5-6. Precondition Settings

3. Click **OK** twice to return to the Sequence File window.
4. Save the sequence.
5. Select **Execute»Single Pass** to execute the sequence.
6. Select the Video test to fail in the Test Simulator dialog box.
7. When the report generates, scroll down the report and observe that the Video test is reported as having failed. Continue scrolling down and notice the status of the Video Diagnostics step. The measurement returned should be between 5 and 15.
8. Save the sequence file and close the Sequence File window when you finish.

End of Part B

End of Exercise 5-1B

Exercise 5-2A Debugging Tests Using LabVIEW

Objective: To debug code modules compiled with LabVIEW.

While debugging a TestStand sequence, you can step into specific code modules and use the debugging tools that are part of the development environment used to create the code module. TestStand can step directly into LabVIEW and LabWindows/CVI code modules. This exercise demonstrates how to step into a LabVIEW code module and use LabVIEW debugging tools to step through the LabVIEW VI.

1. Select **File»Open** and open the `Ex 5-2A Debugging Tests Using LabVIEW.seq` file located in the `C:\Exercises\TestStand I` directory.

   This sequence is the solution created for Exercise 5-1A. Solutions are located in the `C:\Solutions\TestStand I` directory.

2. Click to the left of the `Video Diagnostics` step to place a breakpoint at the step. A stop icon appears, as shown in Figure 5-1.

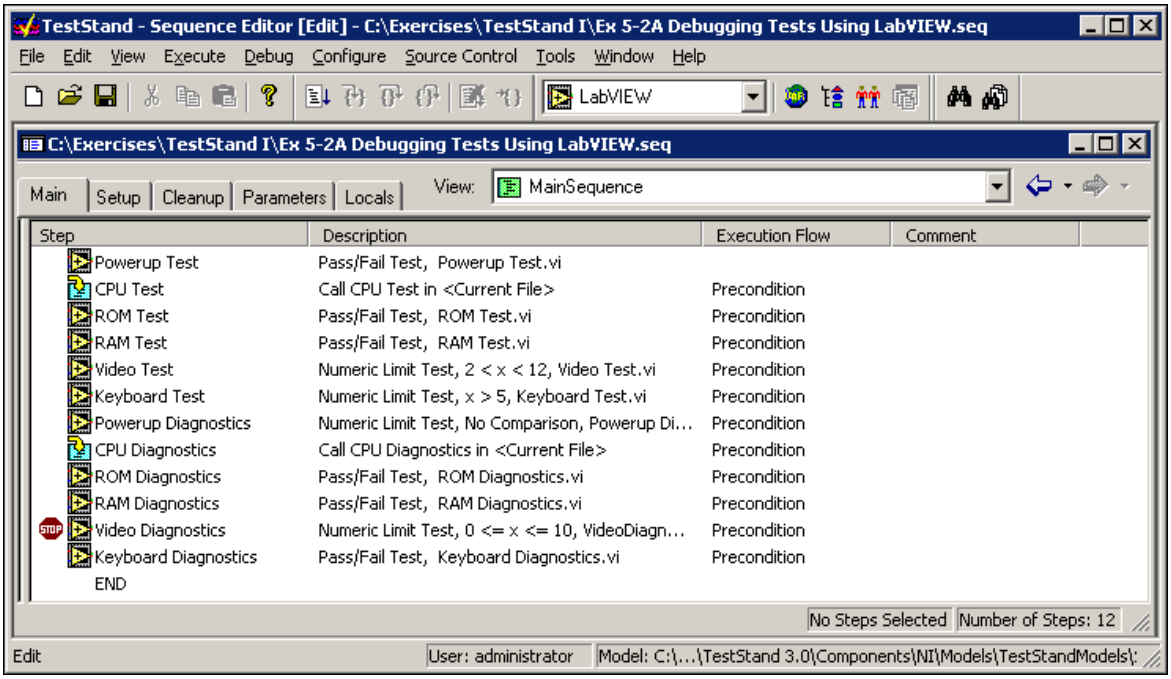


Figure 5-1. Illustrated Breakpoint

3. Execute the sequence and select the Video test to fail in the Test Simulator dialog box. The execution pauses at the breakpoint you inserted.

4. Click the **Context** tab and insert the variable, `RunState.Sequence.Main["Video Diagnostics"].Result.Numeric` into the Watch Expression pane. Figure 5-2 shows the completed Watch Expression pane.

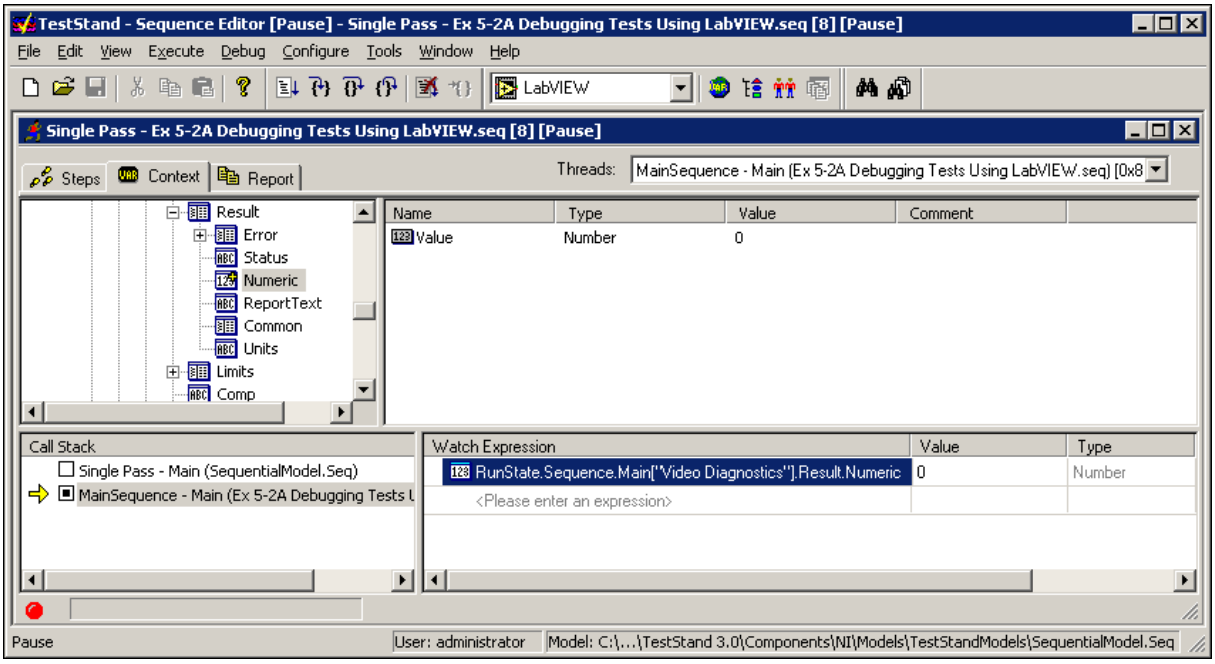


Figure 5-2. Watch Expression Pane

5. Click **Step Into** on the toolbar to step into the LabVIEW code module. LabVIEW launches and displays the front panel for the Video Diagnostics VI code module.
6. Select **Window»Show Block Diagram** from the Video Diagnostics front panel.

7. Select **Window»Show Tools Palette** and select the Breakpoint tool, shown at left, on the **Tools** palette.
8. Using the Breakpoint tool, set a breakpoint on the output of the Multiply function.

9. Click **Run** on the toolbar, shown at left, to run the VI. To debug LabVIEW code, you can use the LabVIEW debugging tools.

10. Click **Pause/Continue**, shown at left, to finish the execution.
11. Click **Return to Caller**, shown at left, to return execution to TestStand.

    The value in the Watch Expression pane should contain the random number that the test generated. This value changes only when the LabVIEW code finishes executing and passes the return values from the **Test Data** cluster back to TestStand.
12. In TestStand, click **Resume** to finish the execution.

Optional: Debugging a LabVIEW DLL

You can only debug a LabVIEW DLL from within the same process. This means that you must debug the LabVIEW DLL using the LabVIEW Operator Interface and not the sequence editor.

To debug a VI that was compiled into a function in a LabVIEW DLL, complete the following steps:

1. In LabVIEW, open the VI that was used to build the function you want to debug in the LabVIEW DLL.
2. Place a breakpoint within this VI.
3. In LabVIEW, open and run the top-level VI of the LabVIEW Operator Interface found at `<TestStand>\OperatorInterfaces\VI\Full-Featured\LabVIEW\TestExec.llb\Full OI - Top-LevelVI.vi`.
4. In the LabVIEW Operator Interface, select the test sequence that calls the function in the LabVIEW DLL that you want to debug.
5. Execute the test sequence from the LabVIEW Operator Interface.

   The execution automatically stops for debugging at the breakpoint that you placed within the VI in step 1.

End of Exercise 5-2A

Exercise 5-2B Debugging Tests Using LabWindows/CVI

Objective: To debug code modules compiled with LabWindows/CVI.

While debugging a TestStand sequence, you can step into specific code modules and use the debugging tools that are part of that development environment. TestStand can step directly into LabVIEW and LabWindows/CVI code modules. This exercise shows how to step into a LabWindows/CVI code module and use LabWindows/CVI debugging tools to step through the code.

1. In the sequence editor, select **File»Open** and navigate to `C:\Exercises\TestStand I\Ex 5-2B Debugging Tests with CVI.seq`.

   This sequence is the solution for Exercise 5-1B. Solutions are located in the `C:\Solutions\TestStand I` directory.

2. Complete the following steps to enable the LabWindows/CVI Adapter to execute steps in an external instance of LabWindows/CVI.
   a. Select **Configure»Adapters**.
   b. Select the **LabWindows/CVI Adapter**.
   c. Click **Configure**.
   d. Enable the **Execute Steps in an External Instance of CVI** option as shown in Figure 5-1.

Note You must have LabWindows/CVI 6.0 or later installed to use the Execute Steps in an External Instance of CVI option.

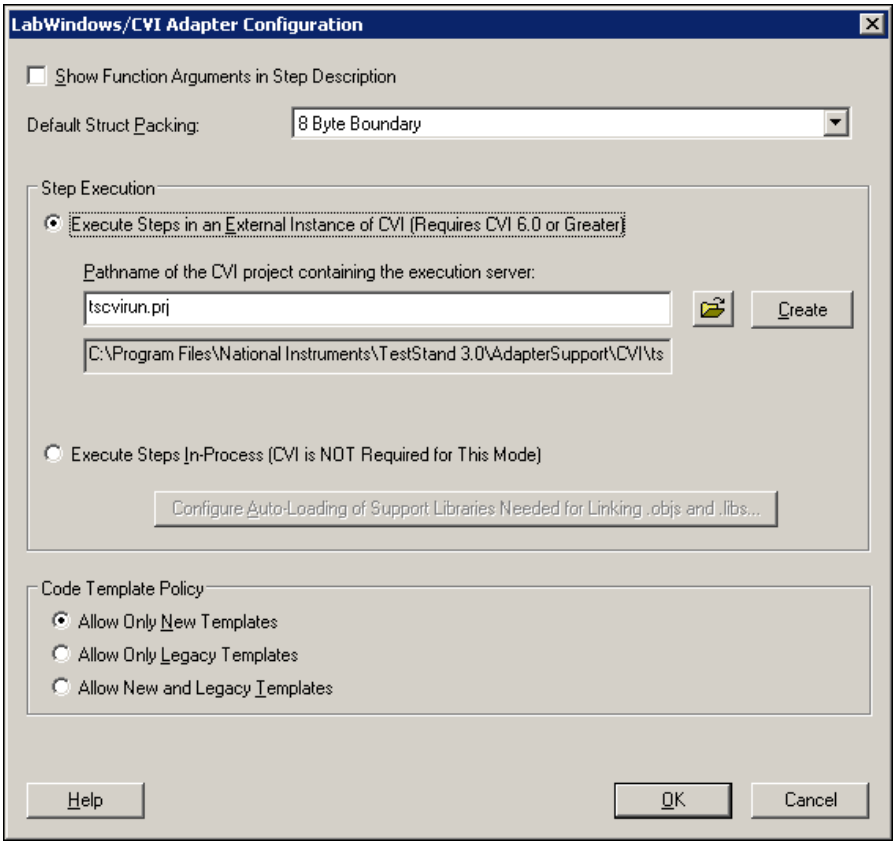


Figure 5-1. LabWindows/CVI Adapter Configuration Dialog Box

Additional Information

If you want to debug the LabWindows/CVI test code, TestStand must execute the LabWindows/CVI code externally, within the LabWindows/CVI ADE. For optimum performance after debugging, set TestStand to run the code in process if the code is compiled as an object file or DLL. Click **Help** on the LabWindows/CVI Adapter Configuration dialog box for a complete description of the options on this dialog box.

3. Click **OK** to close the LabWindows/CVI Adapter Configuration dialog box.
4. Click **OK** to unload all modules and click **Done** in the Adapter Configuration dialog box.
5. Click to the left of the `Video Diagnostics` step to place a breakpoint at the step.
6. Execute the sequence and select the Video test to fail in the Test Simulator dialog box. The execution pauses at the breakpoint you inserted.
7. Click **Step Into** on the toolbar to step into the code module.

TestStand launches the LabWindows/CVI Debugging window, where you can use the LabWindows/CVI stepping tools, Watch window, and Variable window to debug code.

8. When you finish with the LabWindows/CVI Debugging window, click **Go** to finish executing the function and return execution to TestStand.
9. Reset the adapter configuration option to run **LabWindows/CVI Tests In Process**.
   a. Select **Configure»Adapters**.
   b. In the Adapter Configuration dialog box, select the **LabWindows/CVI Adapter** and click **Configure**.
   c. Select the **Execute Steps In Process (CVI is NOT Required for This Mode)** option and click **OK**.
   d. Click **OK** to unload all modules.
   e. Click **Done** to close the LabWindows/CVI Adapter Configuration dialog box.
10. Save and close the sequence file when you finish.

End of Exercise 5-2B

Exercise 5-3A Using the ActiveX API in LabVIEW Code Modules

Objective: To use the ActiveX API to pass data from your code module to TestStand.

This exercise uses the ActiveX Application Programming Interface (API) to pass a string value from TestStand to the LabVIEW VI and write a numeric value and a string value from the code module to TestStand, depending on the value of a randomly generated number. You can use the ActiveX method for passing data to get or set any TestStand data, including local or global variables.

Part A: Creating the Code Module

1. In the sequence editor, open the `Ex 5-3A Using the ActiveX API with LabVIEW.seq` file located in the `C:\Exercises\TestStand I` directory.

   The rest of this exercise describes how to create the code module to be used by this step using ActiveX calls.

2. Right-click the `Video Diagnostics` step and select **Specify Module** from the context menu to open the Edit LabVIEW VI Call dialog box.

3. Click **Create VI** to make a new VI.

4. Navigate to `C:\Exercises\TestStand I\VIs` for **Directory History**. Enter `VideoDiagnosticsActiveX.vi` for the filename. Click **OK**.

   If prompted to resolve the path to the file, select the **Use a relative path for the file you selected** option and click **OK**.

   Additional Information
   TestStand creates the VideoDiagnosticsActiveX VI, containing the **Numeric Measurement** indicator, **Report Text** indicator, and **error out** cluster. Figure 5-1 shows the front panel of the VI.

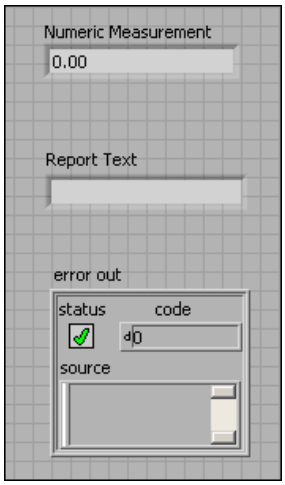


Figure 5-1. VideoDiagnosticsActiveX VI Front Panel

5. Right-click and select the Sequence Context refnum from the **Controls» All Controls»TestStand»Legacy** palette and place it on the front panel.

   The sequence context refnum is an ActiveX Automation Refnum. In this exercise it links the VI to the currently running instance of TestStand. By using the TestStand ActiveX API, you can access steps, sequence properties, and variables.

Tip Click **Search** on the **Controls** palette toolbar to display the **Search** dialog box. Enter `sequence context` in the text box. LabVIEW searches as you type and displays any matches in the search results text box. Click a search result and drag it to the front panel or block diagram to place the object. You can also double-click a search result to highlight its location on the palette.

6. Assign the Sequence Context refnum to a terminal on the connector pane.
   a. Right-click the VI icon in the upper right corner of the front panel and select **Show Connector** from the context menu to display the connector pane for the VI.
   b. Click the upper left terminal of the connector pane, then click the **Sequence Context refnum** to assign the refnum to the terminal on the connector.
7. Delete the **Report Text** and **Numeric Measurement** indicators from the front panel. You will write these values to TestStand using ActiveX instead of passing the values through the terminals of the VI.

End of Part A

Part B: Creating the Code in the Diagram

Typically, ActiveX calls get and set TestStand data values that would be too complex to pass using the VI connector pane. This exercise passes simple information for demonstration purposes. The end result is the same as Exercise 5-1, where you passed the Numeric Measurement and Report Text values using the terminals on the connector pane. In this exercise you read a string value from a TestStand local variable and pass a random number to the `Step.Result.Numeric` property of the step that calls this code module. You also write the string value to the `Step.Result.ReportText` property of the same step, depending on the value of the random number.

You will use two TestStand VIs located on the **TestStand** palette to set and get the step properties. These polymorphic VIs can set values of type Boolean, Boolean Array, Number, Numeric Array, Object, String, or String Array.

- TestStand - Set Property Value (Number) VI—Sets the numeric value of the SequenceContext property the **Lookup String** input specifies, based on the data type wired to the VI.

- TestStand - Get Property Value (String) VI—Writes a value to the SequenceContext property the **Lookup String** input specifies, based on the data type wired to the VI.

In this exercise, a random number passes to the **New Value** input of the TestStand - Set Property Value (Number) VI, which passes that value to the property specified by the **Lookup String** input.

The random number is compared to 10; if it is greater than 10, a text message is read from a TestStand local variable using the TestStand - Get Property Value VI and sent back to TestStand by the TestStand - Set Property Value VI. The False case of the Case structure passes an empty string.

1. Select **Window»Show Block Diagram** and build the block diagram shown in Figures 5-2 and 5-3.

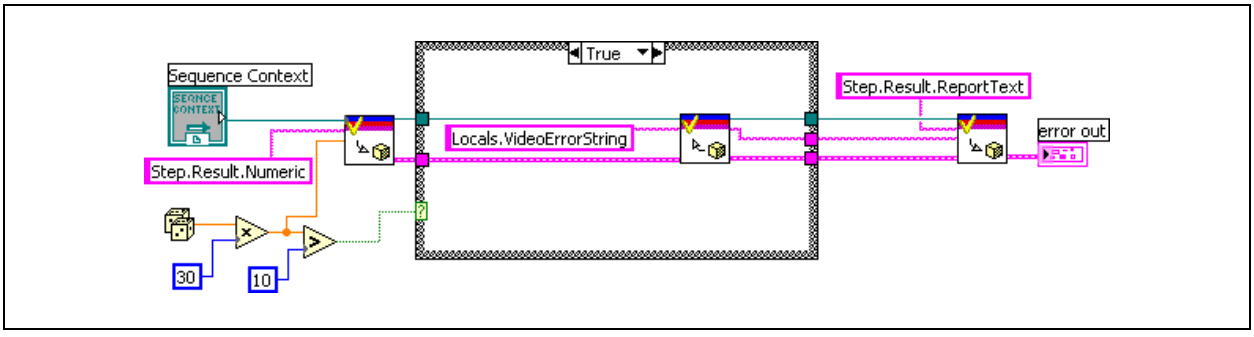


Figure 5-2. True Case Structure

Note When you use the TestStand - Get Property Value VI, you must select the type of data the polymorphic VI accepts. In this case, because you want to read a string value, right-click the VI and select **Select Type»String** from the context menu.

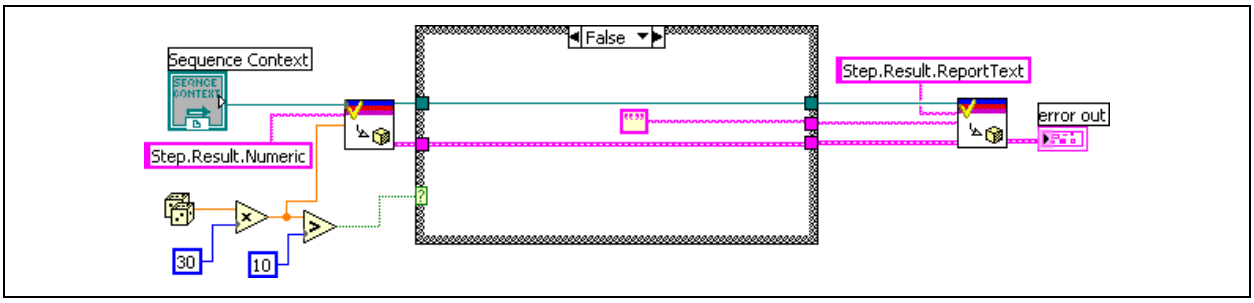


Figure 5-3. False Case Structure

2. Save the VI.
3. Return to the sequence editor. Notice that the `VideoDiagnosticsActiveX.vi` file is now found.
4. Click the **Reload Prototype** button to update the prototype of the VI in the Edit LabVIEW VI Call dialog box. If prompted to keep the current parameter values, click **Yes**. Notice that the new prototype has an additional field for the Sequence Context with a value of ThisContext. ThisContext contains a reference to the currently executing sequence and all related information. The Report Text output and Numeric Measurement output also have been removed because you removed them from the connector pane in LabVIEW. They appear red in the Edit LabVIEW VI Call dialog box to inform you that the fields are not longer included in the prototype.
5. Click **OK** to close the Edit LabVIEW VI Call dialog box. The Video Diagnostics step should appear between the RAM Diagnostics and Keyboard Diagnostics steps.

End of Part B

Part C: Configuring the Step Properties

Add the precondition to the step. The Video Diagnostics step should execute only if the Video Test step fails and the Powerup Test and CPU Test steps pass.

1. Double-click the `Video Diagnostics` step to open the Step Properties dialog box.
2. Click **Preconditions** and set the following preconditions.
   a. Select the **Powerup Test** in the **Insert Step Status** section and click **Insert Step Pass**.

b. Select the **CPU Test** and click **Insert Step Pass**.

c. When prompted to **InsertAllOf** or **InsertAnyOf**, select **InsertAllOf**.

d. Select the **Video Test** and click **Insert Step Fail**.

Figure 5-4 shows the completed Preconditions dialog box.

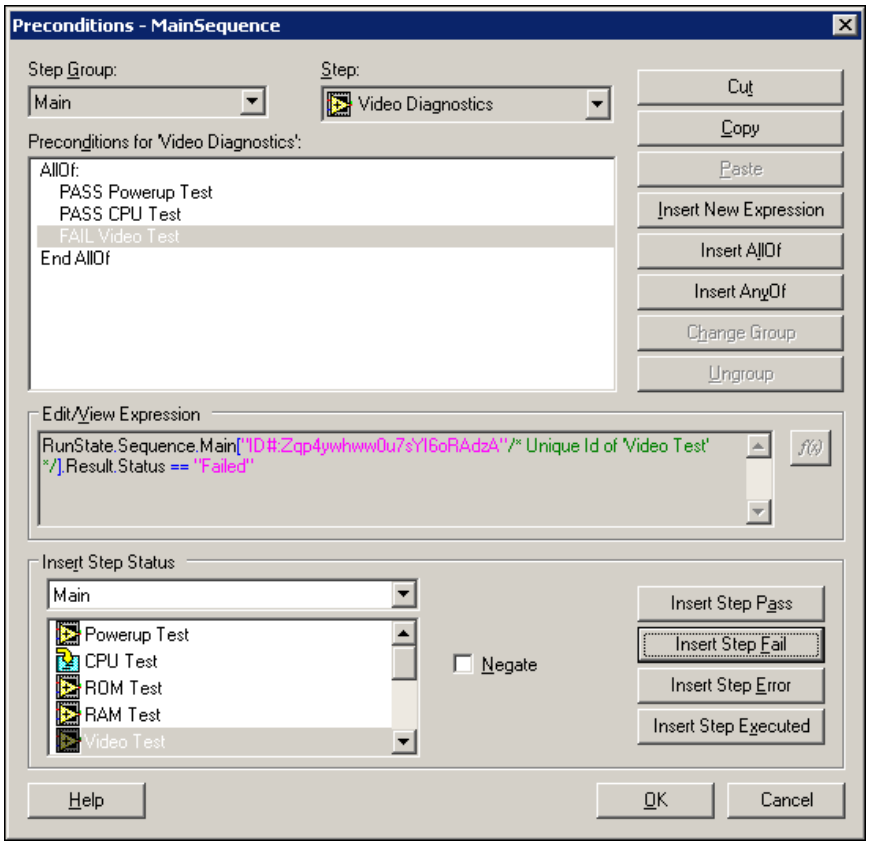


Figure 5-4. Completed Preconditions Dialog Box

3. Click **OK** twice to close the Preconditions and Step Properties dialog boxes.
4. Save the sequence.
5. Execute the sequence by selecting **Execute»Single Pass**.
6. Select the Video test to fail in the Test Simulator dialog box.
7. When the report generates, scroll down the report and observe the status of the Video Diagnostics step.

   The measurement value should fall between 0 and 30. Observe the Report Text string returned from the code module if the value is greater than 10.

End of Exercise 5-3A

Exercise 5-3B Using the ActiveX API in LabWindows/CVI Code Modules

Objective: To use the ActiveX API to pass data from your code module to TestStand.

This exercise uses the ActiveX Application Programming Interface (API) to get a string value from TestStand and pass it to the LabWindows/CVI code module, then write a numeric value and a string value from the code module to TestStand, depending on the value of a randomly generated number. You can use the ActiveX method for passing data to get or set any TestStand data, including local or global variables.

Part A: Creating the Code Module

1. In the sequence editor, open the `Ex 5-3B Using the ActiveX API with CVI.seq` file located in the `C:\Exercises\TestStand I` directory.

   The rest of this lesson describes how to create the code module to be used by this step using ActiveX calls.

2. Right-click the `Video Diagnostics` step and select **Specify Module** from the context menu to open the Edit LabWindows/CVI Module Call dialog box.

3. Configure the **Module** tab to create a function prototype.

   a. **Module Type**: `Dynamic Link Library (*.dll)`
   Module Pathname: `CVI Project Files\Ex5-3VideoDiagnostics.dll`
   Function Name: `VideoDiagnostics`

   Click **New** to create new parameters. Configure the parameters with the following settings. Notice that **Parameter Details**, on the right pane of the dialog box, update to represent the parameter you select from the Parameters table on the left pane of the dialog box.

   b. **Parameter Name**: `seqContextCVI`
   Value: `ThisContext`

   Parameter Details:

   Category: `Object`
   Type: `CVI ActiveX Automation Handle`
   Pass: `By Value`

   c. **Name**: `errorOccurred`
   Value: `Step.Result.Error.Occurred`

   Parameter Details:

   Category: `Numeric`
   Type: `Signed 16-bit Integer`

Pass: `By Reference (by pointer)`
Result Action: `No Action`
Set Error.Code: `unchecked`

d. **Name**: `errorCode`
Value: `Step.Result.Error.Code`

Parameter Details:

Category: `Numeric`
Type: `Signed 32-bit Integer`
Pass: `By Reference (by pointer)`
Result Action: `No Action`
Set Error.Code: `Unchecked`

e. **Name**: `errorMsg`
Value: `Step.Result.Error.Msg`

Parameter Details:

Category: `String`
Type: `C String Buffer`
Pass: `1024`

Figure 5-1 shows the completed **Module** tab.

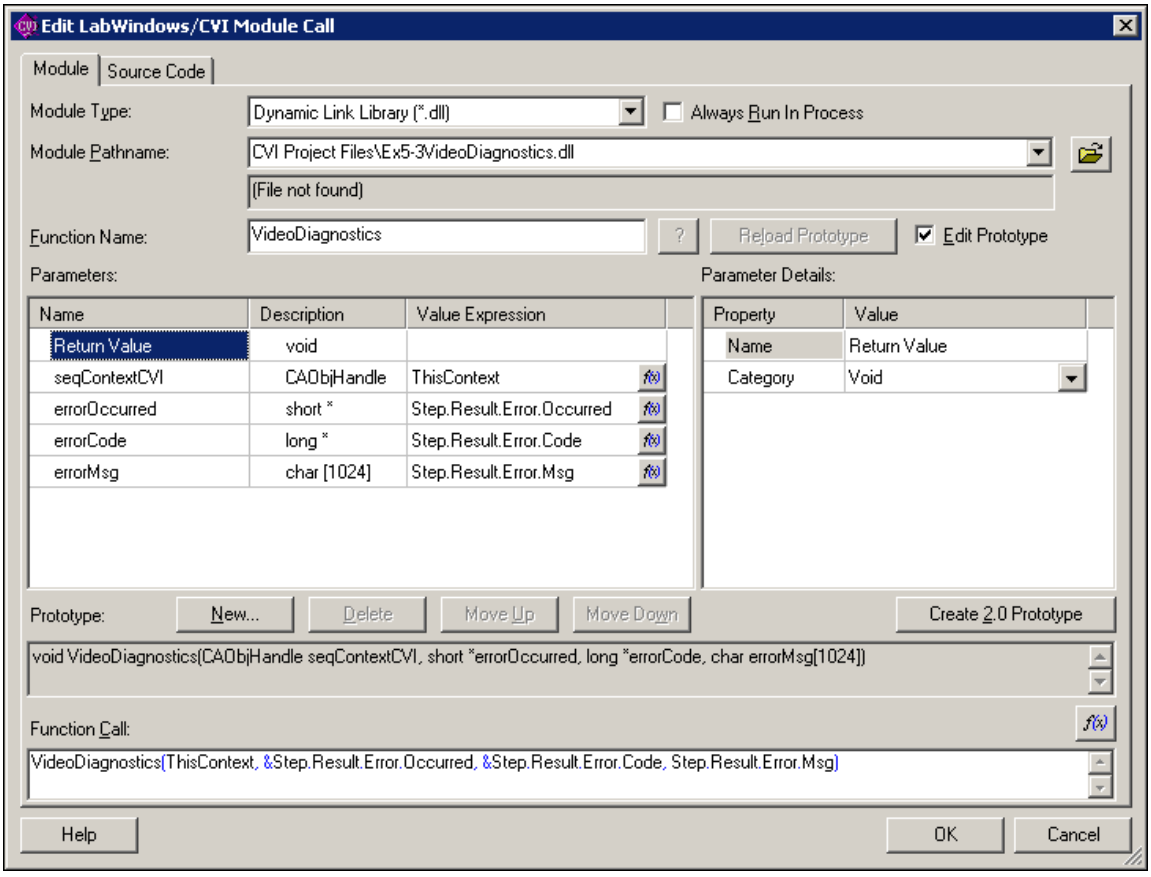


Figure 5-1. Module Tab of Edit LabWindows/CVI Module Call Dialog Box

4. Click the **Source Code** tab and configure the tab as shown in Figure 5-2.

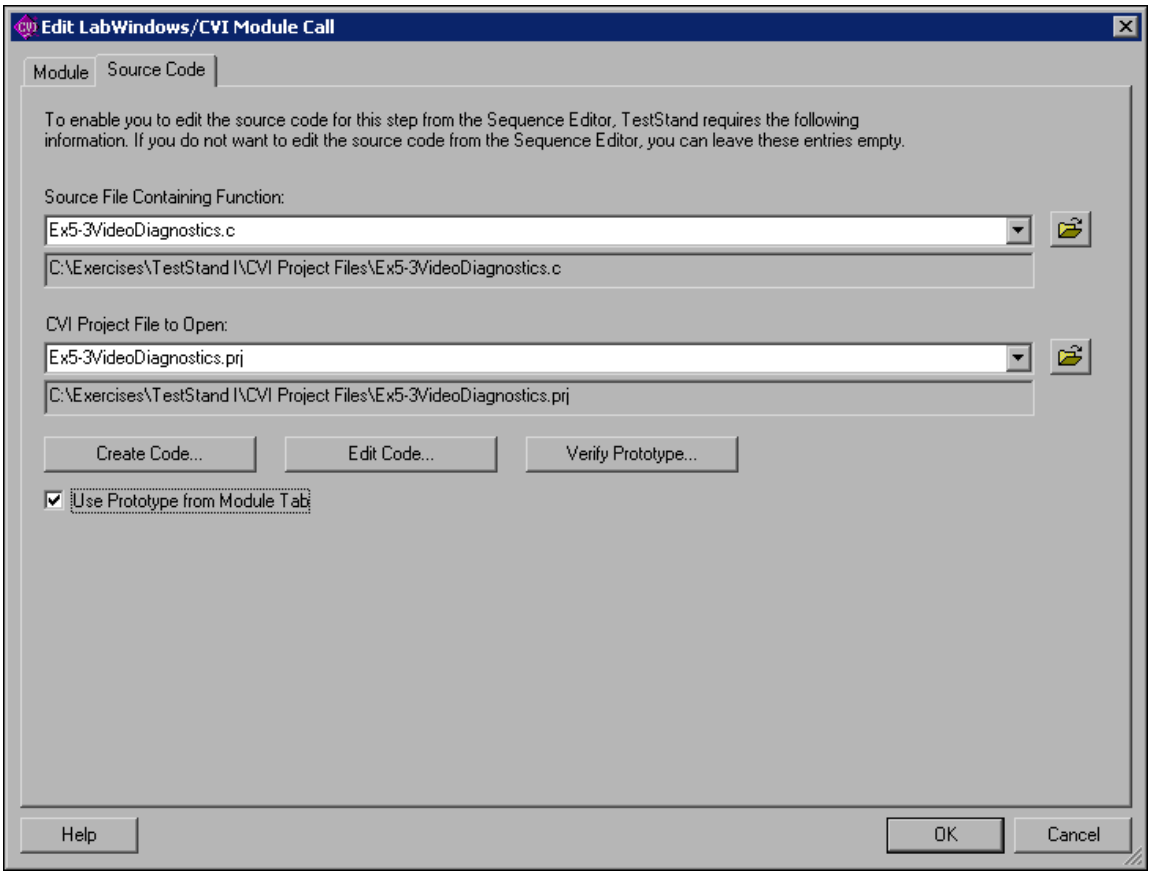


Figure 5-2. Source Tab in Edit LabWindows/CVI Module Call Dialog Box

5. Click **Create Code** to build a new CVI project and source code. Select the `C:\Exercises\TestStand I\CVI Project Files` directory for the project directory and the source directory.
6. If a Prototypes Conflict dialog box displays, select the **Use the Prototype From the Module Tab** option to use the function prototype you created in step 3. Click **OK**.

Note If you already have the source code and DLL created within a project, you can click the **Verify Prototype** button on the **Source Code** tab of the Edit LabWindows/CVI Module Call dialog box. TestStand automatically loads the parameters and you can assign value expressions to the parameters on the **Module** tab.

TestStand launches LabWindows/CVI and opens the `Ex5-3VideoDiagnostics` project and source code containing the `VideoDiagnostics` function located in the `C:\Exercises\TestStand I\CVI Project Files` directory.

7. In the `Ex 5-3 VideoDiagnostics.c` file, complete the following steps to add the ActiveX API code.

   a. Insert the following lines below the `int error = 0` declaration to declare and initialize the variables required to store the random measurement value, TestStand string, and error handling.

```
int error = 0;
ErrMsg errMsg = {'0'};
ERRORINFO errorInfo;
double measurement = 0;
char *videoErrorMessage;
measurement = 5 + (rand()%11);
```

   b. Leave the cursor on a blank line after the last line of code from step a.

   c. Select **Instrument»TestStand API 3.5** to select the TestStand API 3.5 instrument driver. If this driver is not available from the **Instrument** menu, select **Instrument»Load** and load the `tsapicvi.fp` file located in the `<TestStand>\API\CVI` directory.

   d. In the **Select Function Panel** dialog box, select **Get String Value** from the **Property Object»Values** tree, as shown in Figure 5-3.

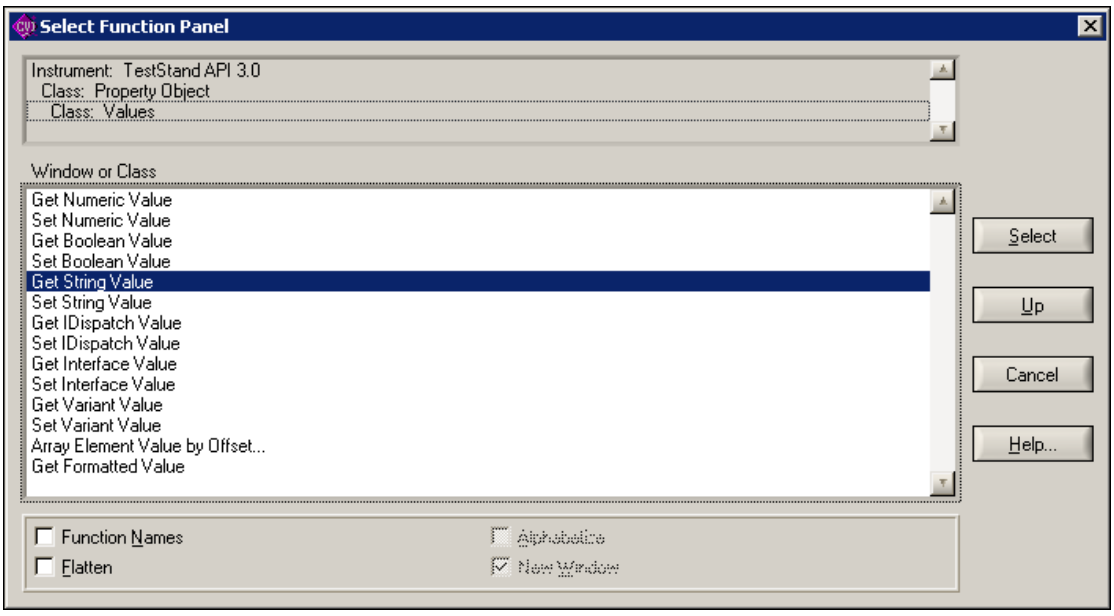


Figure 5-3. Select Function Panel Dialog Box

   e. Click **Select**.

f. Enter the following values in the **Get String Value** dialog box. Figure 5-4 shows the completed dialog.

Object Handle: `seqContextCVI`

Error Info: `&errorInfo`

Lookup String: `"Locals.VideoErrorString"`

Options: `0`

New Numeric Value: `&videoErrorMessage`

Tip The function call updates in the bottom of the Function Panel window as you type in the fields. Use the function call information to see how to complete the fields because the screen limitations do not allow all the text to display

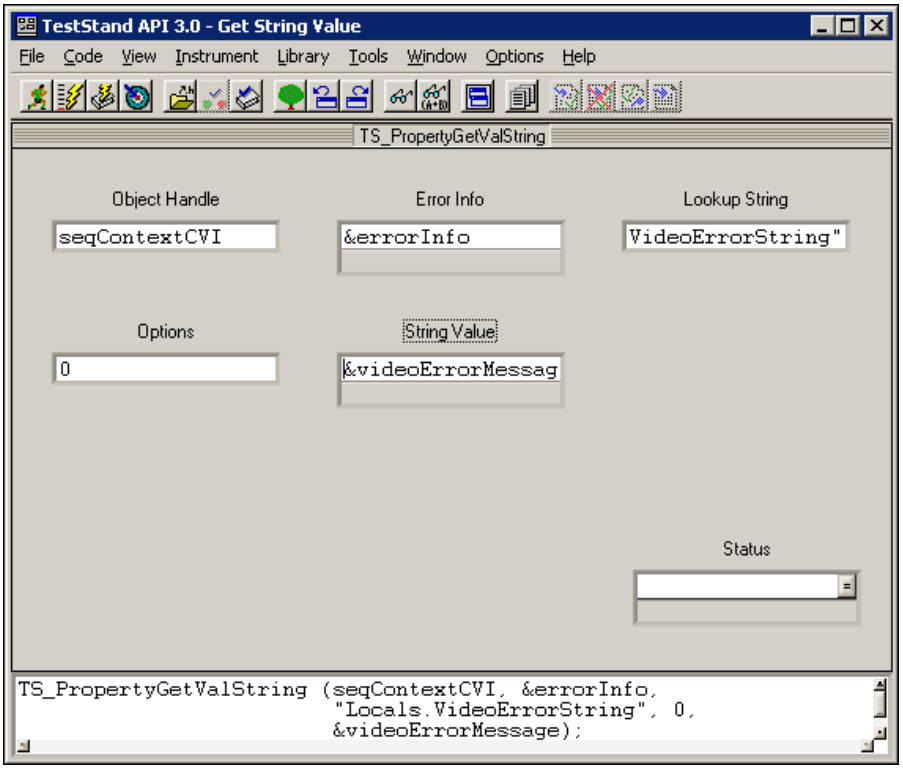


Figure 5-4. Get String Value Dialog Box

Note Right-click in the Function Panel window to display the help for this function. Right-click each control to display help for each parameter.

Additional Information

Following is a brief description of some of the parameters you will use.

- **Object Handle**—A handle to the Sequence Context object in TestStand that passes directly to the function as `struct IDispatch *seqContextCVI`.
- **Error Info**—The returned error information. A C-macro handles this error information and the returned status value. Notice that there are several error status codes defined in the online help.
- **Lookup String**—To specify the property, pass a string that defines a variable or property path based on the object on which you call the method (the object handle). To specify the object itself, pass an empty string (`""`). To specify a subproperty, pass the name of the subproperty. To specify a subproperty of a subproperty, pass a string containing both names separated by a period (`.`). Later in this exercise, you will see how to browse for this parameter.
- **Options**—Refer to the *TestStand Help* for a list of available options.
- **String Value**—The variable to store the value specified by the Lookup String. You will use `videoErrorMessage` for this exercise.

8. From the Function Panel window, select **Code»Set Target File**. Set the **Target File** to `Ex 5-3 VideoDiagnostics.c`. Click the **OK** button.
9. Select **Code»Insert Function Call** to insert this function at the last location of the cursor in the `Ex 5-3 VideoDiagnostics.c` file.
10. Close the function panel.
11. Ensure the cursor is on a blank line after the function call inserted in step 9.
12. Complete the following steps to insert another function call to set the value of the measurement within TestStand.
    a. Select **Instrument»TestStand API 3.5** to select the TestStand API 3.5 instrument driver.
    b. In the Select Function Panel dialog box, select **Set Numeric Value** from the **Property Object»Values** tree to display the Set Numeric Value dialog box.
    c. Click **Select**.
    d. Use the following information to complete the Set Numeric Value dialog box.

Object Handle: `seqContextCVI`

Error Info: `&errorInfo`

Lookup String: `"Step.Result.Numeric"`

Options: `0`

New Numeric Value: `measurement`

Tip The function call updates in the bottom of the Function Panel window as you type in the fields. Use the function call information to see how to complete the fields because the screen limitations do not allow all the text to display.

Note Right-click in the Function Panel window to display the help for this function. Right-click each control to display help for each parameter.

Additional Information

Following is a brief description of some of the parameters you will use.

- **Object Handle**—A handle to the Sequence Context object in TestStand that passes directly to the function as `struct IDispatch *seqContextCVI`.
- **Error Info**—The returned error information. A C-macro handles this error information and the returned status value. Notice that there are several error status codes defined in the online help.
- **Lookup String**—To specify a property, pass a string that defines a variable or property path based on the object on which you call the method (the object handle). To specify the object itself, pass an empty string (`""`). To specify a subproperty, pass the name of the subproperty. To specify a subproperty of a subproperty, pass a string containing both names separated by a period (`.`). Later in the exercise, you will see how to browse for a parameter.
- **Options**—Refer to the *TestStand Help* for a list of available options.
- **New Numeric Value**—The value to write to the value specified by the Lookup String. You use `measurement` for this exercise.

13. Select **Code»Insert Function Call** to insert this function at the last location of the cursor in the `Ex5-3VideoDiagnostics.c` file.
14. Close the function panel.
15. Ensure the cursor is on a blank line after the function call inserted in step 13.
16. Insert another function call to set the text added to the TestStand report.
    a. Select the **Instrument»TestStand API 3.5** instrument driver.
    b. In the **Select Function Panel** dialog box, select **Set String Value** from the **Property Object»Values** tree to display the **Set String Value** function panel.

c. Use the following information to complete the Set String Value function panel.

Object Handle: `seqContextCVI`

Error Info: `&errorInfo`

Lookup String: `"Step.Result.ReportText"`

Options: `0`

New String Value: `videoErrorMessage`

Tip The function call updates in the bottom of the Function Panel window as you type in the fields. Use the function call information to see how to complete the fields because the screen limitations do not allow all the text to display.

This function writes the test value that you read from the TestStand local variable to the `Step.Result.ReportText` TestStand variable.

17. Select **Code»Insert Function Call** to insert this function at the last location of the cursor in the `Ex5-3VideoDiagnostics.c` file.
18. Return to the `Ex5-3VideoDiagnostics.c` file and make sure the function calls are correct. The following code is for illustration purposes only.

```
TS_PropertyGetValString (seqContextCVI, &errorInfo,
"Locals.VideoErrorString", 0, videoErrorMessage);
TS_PropertySetValNumber (seqContextCVI, &errorInfo,
"Step.Result.Numeric", 0, measurement);
TS_PropertySetValString (seqContextCVI, &errorInfo,
"Step.Result.ReportText", 0, videoErrorMessage);
```

19. Select **File»Save All**.
20. Return to TestStand and click **OK** to close the Edit LabWindows/CVI Module Call dialog box. Do not close LabWindows/CVI.
21. In the sequence editor, right-click the `Video Diagnostics` step and select **Browse Sequence Context** from the context menu to open the Browse Variables and Properties in Sequence Context dialog box.
22. Select `Step.Result.Numeric` as shown in Figure 5-5.

Additional Information

This is the same Lookup String passed into the `TS_PropertySetValNumber` function call. Notice that you could have copied this variable/property with the **Copy Property Path** button and pasted it into the function call. Also notice that any property or variable can be referenced with similar Lookup Strings.

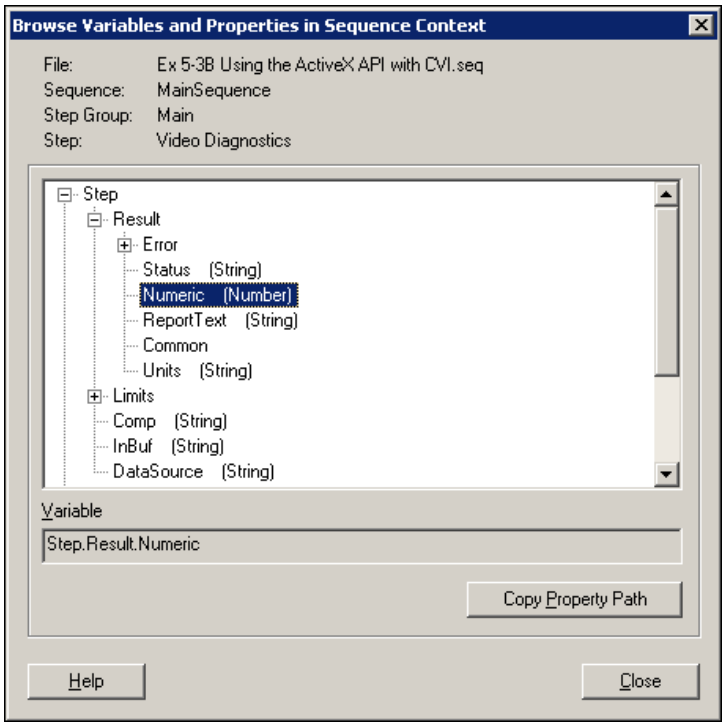


Figure 5-5. Inserting a Variable Using Copy Property Path

23. Click **Close** to close the dialog box.
24. Return to LabWindows/CVI and implement error handling in the code module by entering the additional code shown below in bold.

    The `tsErrChkMsgPopup()` macro is defined in `tsutil.h`. When an error occurs within a function call surrounded by this macro, the macro displays the error description, sets the `error` variable, and jumps to the `Error:` label.

Tip Remember that C compilers such as LabWindows/CVI are case sensitive.

```
#include "stdtst.h"
#include "tsutil.h"
void __declspec(dllexport) VideoDiagnostics(CAObjHandle seqContextCVI,
short *errorOccurred, long *errorCode, char errorMsg[1024])
{
   int error = 0;
   ErrMsg errMsg = {'\0'};
   ERRORINFO errorInfo;
   double measurement = 0;
   char *videoErrorMessage;
   measurement = 5 + (rand()%11);
   tsErrChkMsgPopup (TS_PropertyGetValString
   (seqContextCVI,&errorInfo,"Locals.VideoErrorString", 0,
```

```
   &videoErrorMessage));
   tsErrChkMsgPopup (TS_PropertySetValNumber
   (seqContextCVI,&errorInfo,"Step.Result.Numeric", 0, measurement));
   if (measurement >10) {
   tsErrChkMsgPopup (TS_PropertySetValString (seqContextCVI,
   &errorInfo,"Step.Result.ReportText", 0,videoErrorMessage));
   }
Error:
   // FREE RESOURCES HERE
   // If an error occurred, set the error flag to cause a run-time error
in TestStand
    if (error < 0)
   {
   *errorOccurred = TRUE;
   // OPTIONALLY SET THE ERROR CODE AND STRING
   *errorCode = error;
   strcpy(errorMsg, errMsg);
   }
   return;
}
```

25. Save the source file.
26. Select **Build»Target Type** and set the project target to **Dynamic Link Library**.
27. Select **Build»Configuration»Debug** to configure the DLL to be built as a debuggable version.
28. Select **Build»Create Debuggable Dynamic Link Library** to create the DLL.

    If LabWindows/CVI displays a message that access is denied, you need to unload the DLL. Select **File»Unload All Modules** in the sequence editor so TestStand can overwrite the DLL.
29. A dialog box launches to indicate that the DLL was built successfully. Click **OK** to close the dialog box.
30. Return to the TestStand Sequence Editor.

End of Part A

Part B: Configuring the Step Properties

Add the Preconditions to the step. The Video Diagnostics step should execute only if the Video Test step fails and the Powerup Test and CPU Test steps pass.

1. Double-click the `Video Diagnostics` step to open the Step Properties dialog box.
2. Click **Preconditions** and set the preconditions.
   a. Select the **Powerup Test** in the **Insert Step Status** section and click **Insert Step Pass**.
   b. Select the **CPU Test** and click **Insert Step Pass**.
   c. When prompted to **InsertAllOf** or **InsertAnyOf**, select **Insert AllOf**.
   d. Select the **Video Test** and click **Insert Step Fail**.

   Figure 5-6 shows the completed Preconditions dialog box.

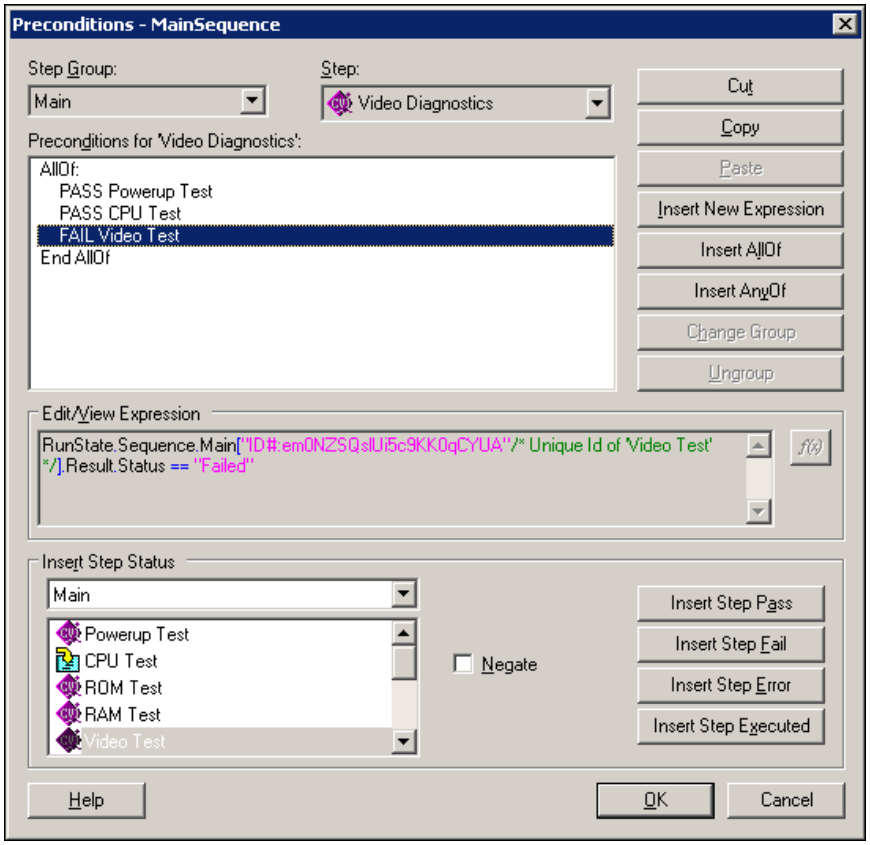


Figure 5-6. Completed Preconditions Dialog Box

3. Click **OK** twice to close the Preconditions and Step Properties dialog boxes.
4. Save the sequence.

5. Execute the sequence by selecting **Execute»Single Pass**.
6. Select the Video test to fail in the Test Simulator dialog box.
7. Verify that the correct data is returned. The measurement from the Video Diagnostics test should be between 5 and 15. Save and close the sequence file when you finish.

End of Exercise 5-3B

Self Review

1. List and describe the methods available to pass data between TestStand and external code modules.

2. How do you debug a DLL using the LabWindows/CVI Adapter?

3. Briefly explain what ActiveX is and how it relates to TestStand.

4. Which adapters allow the use of ActiveX to transfer data?

5. What type of information can you access from code modules using ActiveX?

6. What is a lookup string?

7. What are code templates? Why are they used?

Notes

Notes

6

Importing and Exporting Properties

Lesson 6: Importing and Exporting Properties

In this lesson, you will:

- **Import and export variables and properties for sequences using the Import/Export Properties tool**
- **Use the Property Loader step type to update the properties and variables in a sequence dynamically**

Introduction

This lesson describes how to import and export variables and properties for sequences using the Import/Export Properties tool and the Property Loader step type.

Import/Export Properties Tool

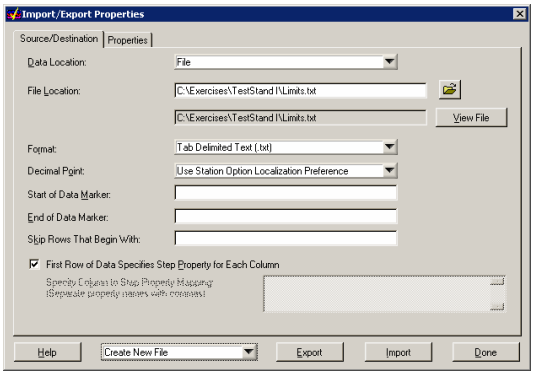
- Using the Import/Export Properties tool
- Select Tools»Import/Export Properties
- You can import and export properties to and from the following formats:
  - File (`*.txt`, `*.csv`, `*.xls`)
  - Database

Import/Export Properties Tool

For some applications, it might be more efficient to operate the same sequences using different variable and property values. For example, testing multiple models within the same product line, rather than writing separate sequences for every model. You can use the Import/Export Properties tool to manually import variable and property values into a sequence from a file, database, or the Windows clipboard. You can also use this tool to export variable and property values from a sequence to a file, database, or the Windows clipboard. Select **Tools»Import/Export Properties** to launch the Import/Export Properties tool.

Import/Export Properties Dialog Box

Dialog box contents depend on the Data Location you specify.



Import/Export Properties Dialog Box

When you select **Tools»Import/Export Properties**, the Import/Export Properties dialog box launches.

The Import/Export Properties dialog box contains the following options when you select **Database** from the **Data Location** ring control.

- **Source/Destination**
- **Properties**
- **Additional Columns**

The **Import/Export Properties** dialog box contains the following options when you select **File** or **Clipboard** from the **Data Location** ring control.

- **Source/Destination**
- **Properties**

The fields on each tab also differ according to the **Data Location**. Refer to the *TestStand Help* for more information about the Import/Export Properties dialog box.

Property Loader Step Type

- **The Property Loader step type dynamically updates properties and variables in a sequence at run time**
- **Place this step in the Setup step group of a sequence**
- **The Property Loader step type uses properties and variables in the following formats:**
  - **File** (`*.txt`, `*.csv`, `*.xls`)
  - **Database**

Property Loader Step Type

Use the Property Loader step to update the properties and variables in a sequence dynamically. In this way, you can develop a common sequence that can test different models of a UUT, where each model requires unique property or variable values.

You usually insert the Property Loader step in the Setup step group of a sequence, so that the Property Loader step initializes the limit values before the steps in the Main step group execute. The source of the values can be a tab-delimited text file (`.txt`), a comma-delimited text file (`.csv`), an Excel file (`.xls`), or a database. You can use the Property Loader step together with expressions to specify different files or database data links dynamically.

To examine how to use the Property Loader step type with databases, load the `PropertyLoader.seq` example located in the `<TestStand>\Examples\Database` directory.

Exercise 6-1: Importing and Exporting Properties

Objective: To use a file or database to edit the properties and variables of a sequence.

Estimated Time: 15 minutes

Refer to page 6-9 for instructions for this exercise.

Exercise 6-2: Using the Property Loader Step Type

Objective: To use the Property Loader step type to dynamically import properties and variables from a file or database.

Estimated Time: 5 minutes

Refer to page 6-16 for instructions for this exercise.

Exercise 6-3: Using Expressions to Load Properties

Objective: To use the Property Loader step type to import different values based on the value of a TestStand expression.

Estimated Time: 15 minutes

Refer to page 6-21 for instructions for this exercise.

Lesson 6: Summary

- **Use the Import/Export Properties tool to import/export sequence properties and variables to a file or database**
- **Launch the tool using Tools»Import/Export Properties**
- **Use the Property Loader step to specify a file or database from which to load properties and variables for a particular sequence execution**
- **Change files or databases dynamically using expressions to specify the source values to be loaded by the Property Loader step**

Summary

You can import or export sequence properties and variables manually using the TestStand Import/Export Properties tool. You can export this information in file (`.txt`, `.csv`, and `.xls`) or database format. You can also import property and variable values during sequence run time using the Property Loader step type. The Property Loader step type allows you to dynamically change the file or database to import the values from using expressions to specify which file or database to use.

Exercise 6-1 Importing and Exporting Properties

Objective: To use a file or database to edit the properties and variables of a sequence.

This exercise demonstrates how to use the TestStand Import/Export Properties tool to import and export properties of various tests in a sequence.

1. Open a new sequence file and save it as `Ex 6-1 Resistor Test.seq` in the `C:\Exercises\TestStand I` directory.
2. Insert a Numeric Limit Test step into the sequence using the **<None> Adapter**.

   Additional Information
   Use the <None> Adapter to create a step in a sequence that does not call any external code. Otherwise, <None> steps behave in the same way as other steps; that is, they create data space for variables and properties based on their step type and make comparisons with that data to determine whether the test passes or fails.
3. Name the step `Resistor 1 Test`.

   Additional Information
   Because this step uses the <None> Adapter, it does not call any code; thus, it does not return a numeric result. Instead, you will place a value in the result field using a post expression.
4. Right-click the `Resistor 1 Test` step and select **Properties** from the context menu.
5. Click the **Expressions** tab and click inside the **Post-Expression** section.
6. Enter the expression, `Step.Result.Numeric = Random(0,1)`, by typing it or using the Expression Browser dialog box, which you can access by clicking the **Expression Browse** button.

   This expression defines the numeric result for this step to be a random number between 0 and 1.
7. Click **OK** to return to the Sequence File window.
8. Select the new step and press the **<Ctrl-C>** keys or right-click it and select **Copy** from the context menu to copy.
9. Paste four copies of the step into the sequence by pressing the **<Ctrl-V>** keys four times or right-clicking in the window and selecting **Paste** from the context menu.
10. Name the steps `Resistor 2 Test`, `Resistor 3 Test`, `Resistor 4 Test`, and `Resistor 5 Test`.

Tip Right-click the step and select **Rename** to rename the step.

11. Click the **Locals** tab in the `MainSequence` and insert a new local variable of the string data type. Name the variable `ResultString`.
12. Click the **Main** tab within the `MainSequence` and add a String Value Test step to the end of the sequence. For this right-click in the Sequence File window and select **Insert Step»Tests»String Value Test** from the context menu.
13. Name the new step `Instrument Query`.
14. Right-click the step and select **Properties** from the context menu, then click the **Expressions** tab.
15. Enter a Post-Expression of `Step.Result.String = Locals.ResultString` for the `Instrument Query` step.
16. Click **OK** to return to the Sequence File window.
17. Save the changes you made to the sequence file. The sequence should appear as shown in Figure 6-1.

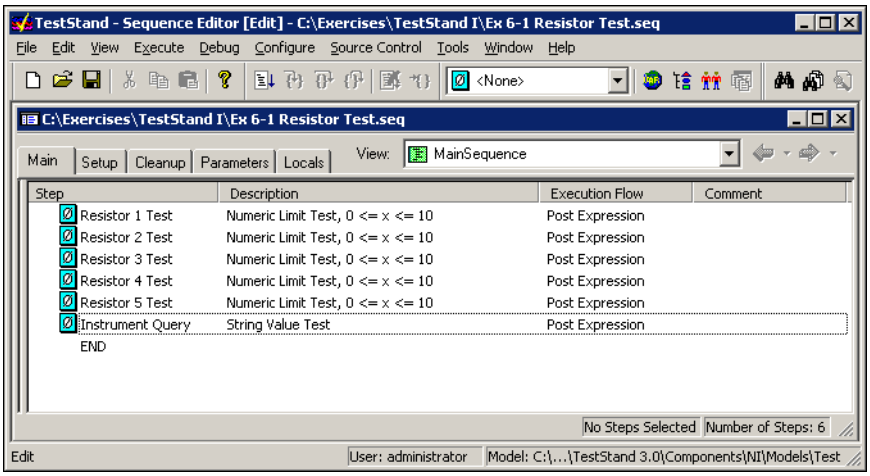


Figure 6-1. Resistor Test Sequence File

18. Select **Tools»Import/Export Properties** from the sequence editor menu to display the Import/Export Properties dialog box.

    When you edit a sequence file, you can use the Import/Export Properties dialog box to import values from a database, file, or clipboard into step properties or variables or to export values from step properties or variables to a database, file, or clipboard. In this exercise, you will use this dialog box to export the selected properties to a file and then import the updated properties to the test sequence.
19. Select the **Source/Destination** tab.

Click **Help** for more information about the options for file, clipboard, and data locations on the Source/Destination tab of the Import/Export Properties dialog box.

a. Select **File** from the **Data Location** ring control.

b. Specify the path, `Limits.txt`, for the **File Location** control.

   After you specify the path, **`File not found`** displays because the file does not exist yet and will be created automatically.

c. Select **Tab Delimited Text (.txt)** from the **Format** ring control.

d. Select **Use Operating System Setting** from the **Decimal Point** ring control.

Leave the other controls on this tab in their default settings. Figure 6-2 shows the resulting Source/Destination tab.

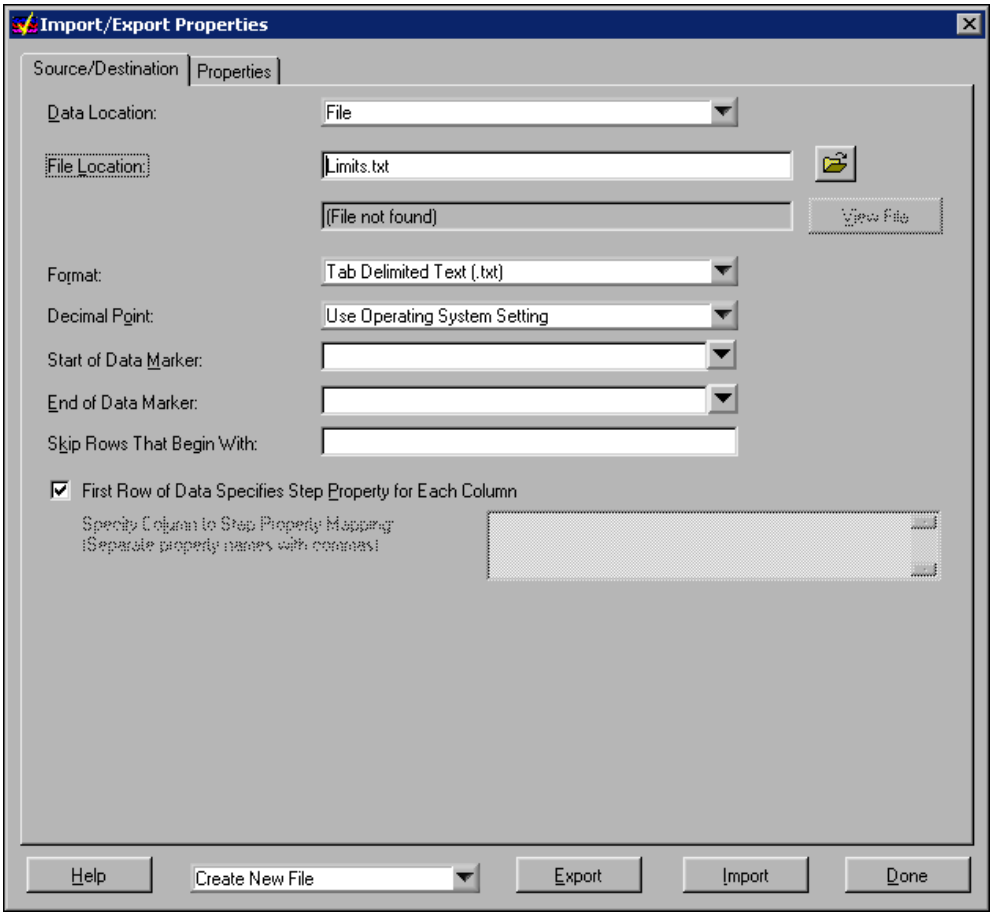


Figure 6-2. Source/Destination Tab

20. Select the **Properties** tab to specify which properties to export.

    For this exercise, you should export all the `Step.Limits` properties and the `Locals.ResultString` property from the `MainSequence`.

    a. Select the `MainSequence` from the **Sequence** ring control.

       Notice the list of available properties. Select the properties you want from the list.

    b. Click **Limits** and select all the `Step.Limits` properties.

    c. To add the `Locals.ResultString` property to the list of selected properties, highlight the property in the list of available properties and click the **>** button.

    Figure 6-3 shows the resulting Properties tab.

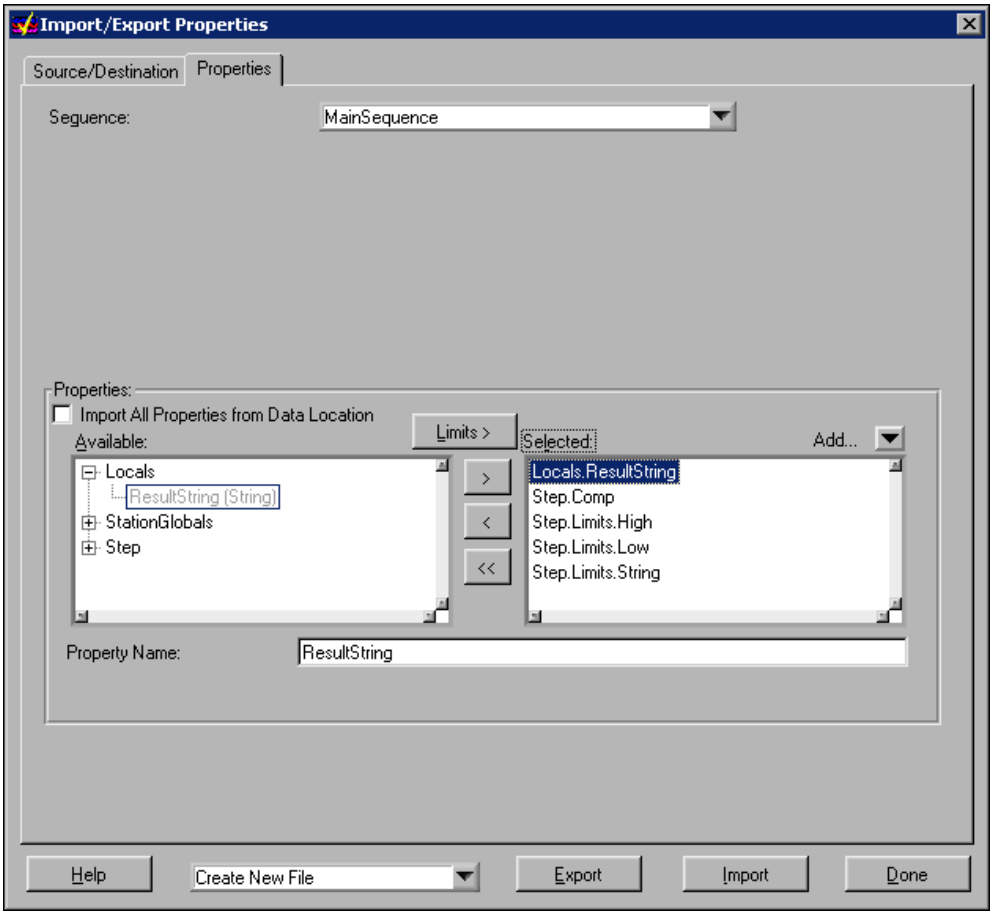


Figure 6-3. Properties Tab

21. Click **Export** on the Import/Export Properties dialog box to export the selected properties and their corresponding values to a file.

    If prompted to select a file path in which to save data, select `Limits.txt` and click **Save**. The Export dialog box should display as shown in Figure 6-4.

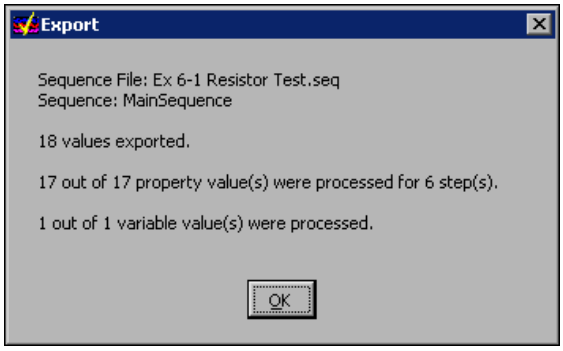


Figure 6-4. Export Dialog Box

This message indicates that you have successfully exported the selected properties to the file `C:\Exercises\TestStand I\Limits.txt`.

22. Click **OK** and close the Export dialog box.
23. Click **Done** to close the Import/Export Properties dialog box.
24. Use Windows Explorer to navigate to the `C:\Exercises\TestStand I` directory and open the `Limits.txt` file. Leave this file open for the next step.

Additional Information

The `Limits.txt` file created by TestStand shows each step containing any of the selected properties occupying a row. In this case, all steps have associated properties, but if there were steps that did not contain the selected properties, these steps would not appear in the file. Each column of the file is used for a property field. The five numeric tests use the `Limits.High` and `Limits.Low` columns, while the string value test uses only the `Limits.String` column. The variable properties use the sections below the steps for each scope of variable: locals, sequence file globals, and station globals. In this case, the only variable exported was the `ResultString` local variable.

25. Edit the text file to reflect the values as shown in Figure 6-5. Save the changes made to the `Limits.txt` file.

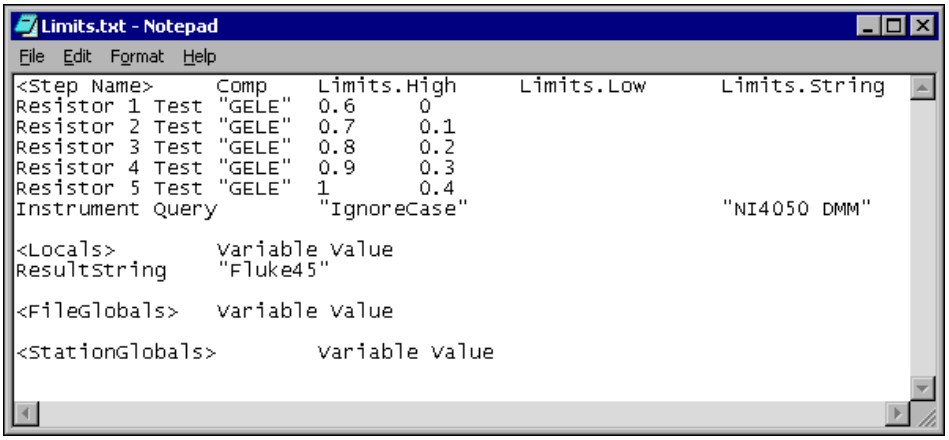


Figure 6-5. Limits.txt File

26. Launch the Import/Export Properties dialog box again, verifying that the correct file and properties are specified.
27. Click **Import** in the Import/Export Properties dialog box. The Import Complete dialog box displays, as shown in Figure 6-6, indicating that 18 limits were found and imported from the file.

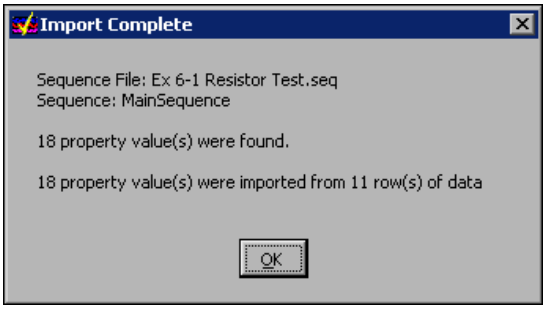


Figure 6-6. Import Complete Dialog Box

28. Click **OK** to close the Import Complete dialog box.
29. Click **Done** to close the Import/Export Properties dialog box.

    Examine the limits of the test steps to verify that the updated values reflect those specified in the file. Your sequence file should be similar to Figure 6-7. Also, verify that the local variable `ResultString` contains the proper updated value.

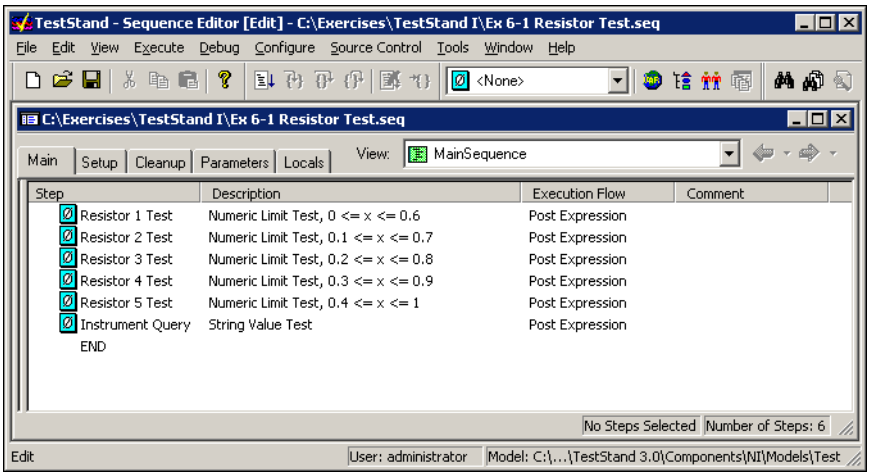


Figure 6-7. Completed Resistor Test Sequence File

30. Save the sequence file.

End of Exercise 6-1

Exercise 6-2 Using the Property Loader Step Type

Objective: To use the Property Loader step type to dynamically import properties and variables from a file or database.

It is often more convenient to load properties automatically from a file or database during the execution of a sequence. You can use the Property Loader step type to dynamically load properties at run time from any point within the test sequence. It is a convenient method because the test operator does not have to manually import the properties; the import is done as a part of the test sequence.

1. Open the `Ex 6-1 Resistor Test.seq` sequence file located in the `C:\Exercises\TestStand I` directory. This is the sequence file you completed in Exercise 6-1.
2. Select **File»Save As** and save the file as `Ex 6-2 Resistor Test with Property Loader.seq` in the `C:\Exercises\TestStand I` directory.
3. Select the **Setup** tab within the `MainSequence`.
4. Right-click and select **Insert Step»Property Loader** from the context menu to insert a Property Loader step in the Setup step group. This allows the Property Loader step to initialize the property and variable values before the steps in the Main step group execute.
5. Name the step `Load Properties`.
6. Right-click the `Load Properties` step and select **Edit Property Loader** from the context menu.

   Additional Information
   The Edit Property Loader dialog box allows you to select which properties you want to load and from what data location. Notice the similarity in the configuration tabs between the Edit Property Loader dialog box and the Import/Export Properties dialog box you used in the Exercise 6-1.
7. Click the **Properties** tab.
8. Select **File** from the **Data Location** ring control.
9. Insert `Step.PropertiesList` for the **Properties List Source** control value. Click **Limits** to specify that you want to load all the step limit properties. Select the `Locals.ResultString` property and click the **>** button to add it to the list of selected properties.

Figure 6-1 shows the resulting Properties tab.

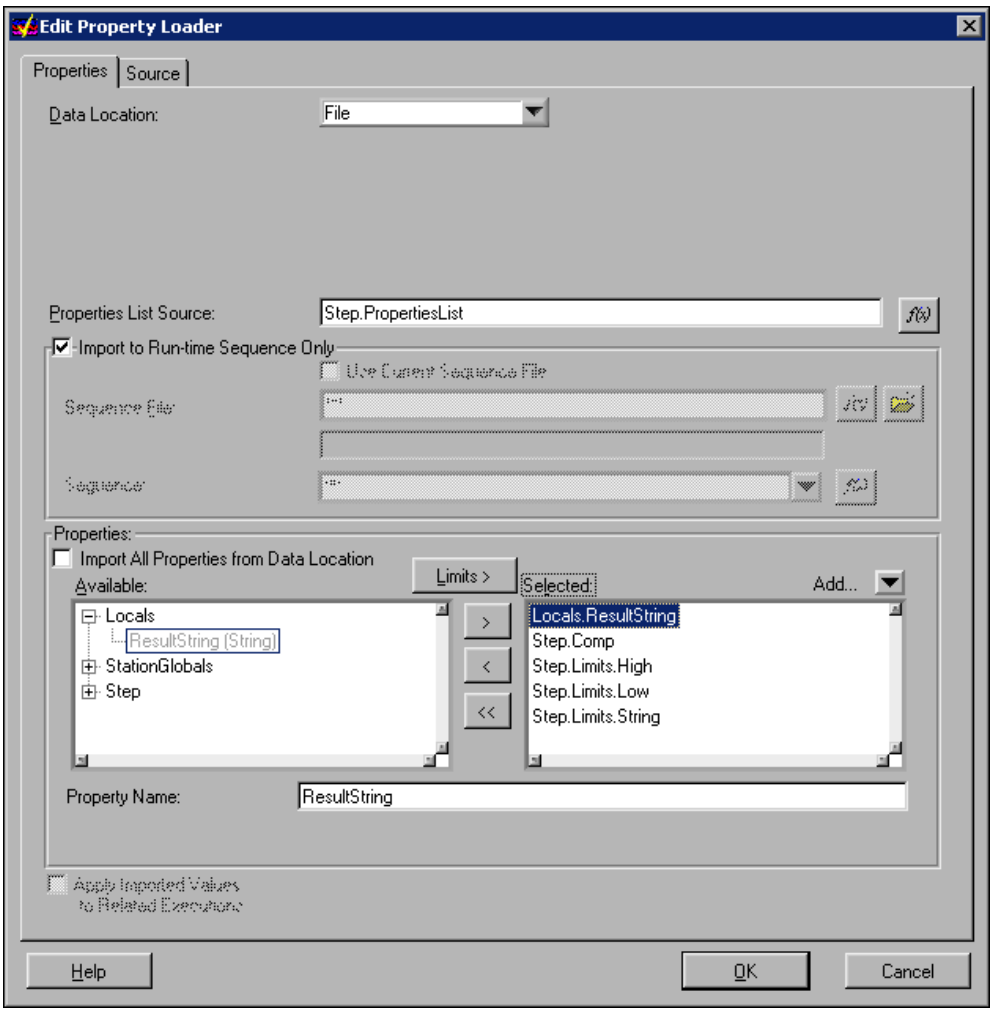


Figure 6-1. Properties Tab in Edit Property Loader Dialog Box

10. Click the **Source** tab.
11. Enable the **Select Specific File** option and click **Browse** in that section. Navigate to the `C:\Exercises\TestStand I\Limits.txt` file.

    If prompted, specify that you want to add the directory containing the selected file to the list of search directories.
12. Select **Tab Delimited Text (.txt)** in the **Format** ring control.
13. Select **Use Operating System Setting** from the **Decimal Point** ring control.

    Leave the other controls on this tab in their default settings. Figure 6-2 shows the resulting Source tab.

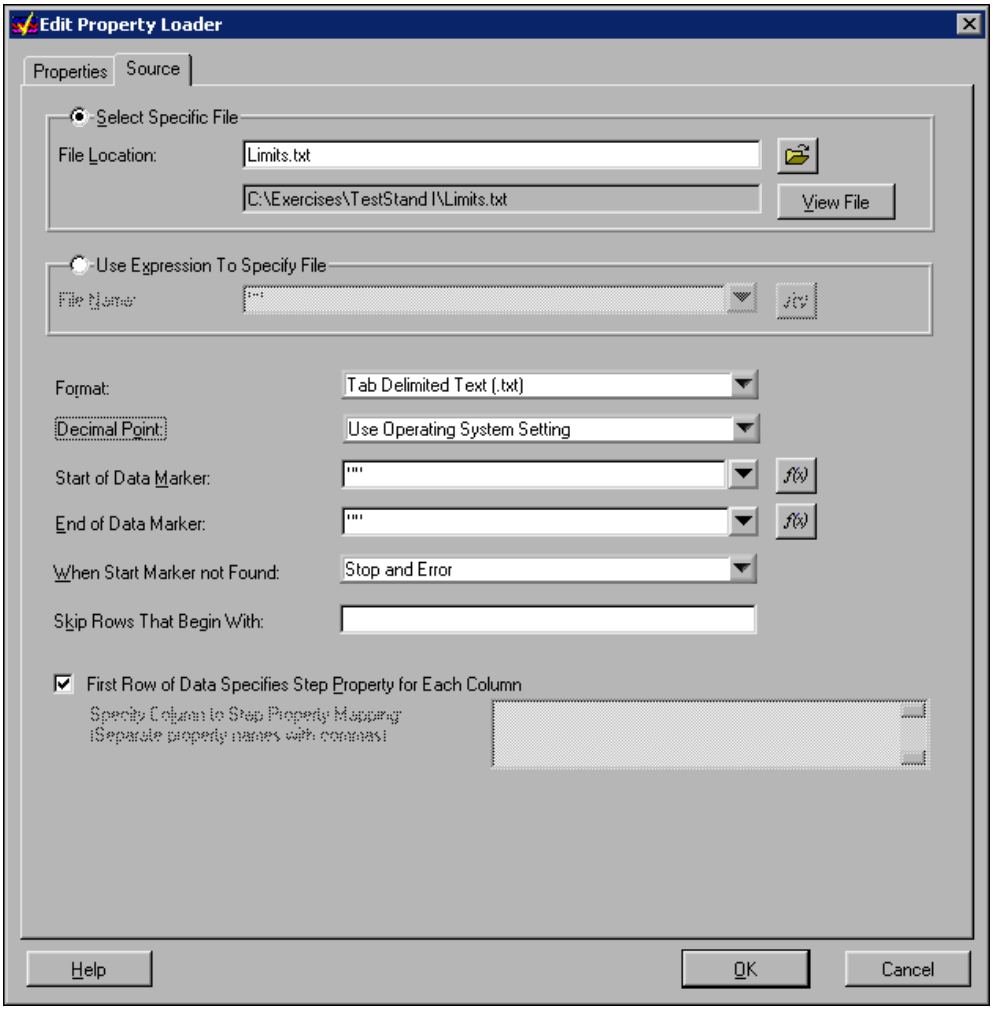


Figure 6-2. Source Tab in the Edit Property Loader Dialog Box

14. Click **OK** to store your changes and close the Edit Property Loader dialog box.
15. Save the sequence file.
16. Open `C:\Exercises\TestStand I\Limits.txt` and edit the property values to reflect the changes shown in Figure 6-3. Save the `Limits.txt` file after making the changes.

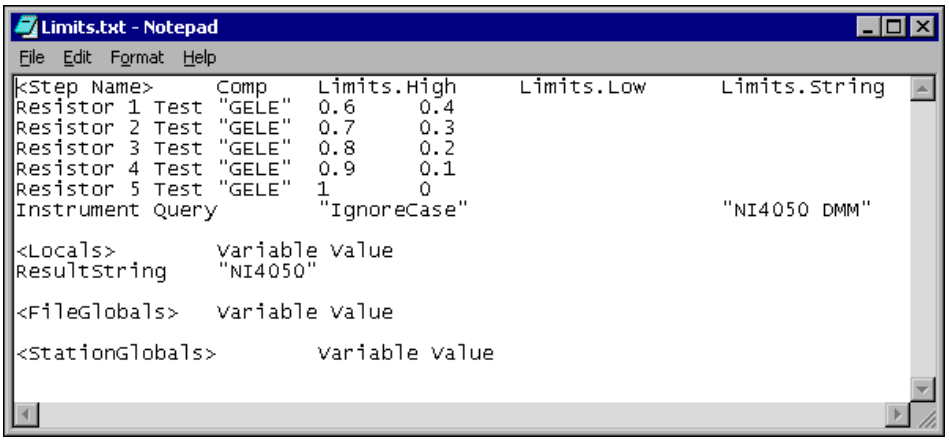


Figure 6-3. Limits.txt File

17. Place a breakpoint at the `Load Properties` step in the Setup step group.
18. Select **Execute»Single Pass** and execute the sequence.
19. When the execution pauses on the `Load Properties` step, click the **Context** tab and drag the low and high limits of the `Resistor 1 Test` from the context menu into the Watch window. The high and low limits are located under `RunState.Sequence.Main.["Resistor 1 Test"]([0]).Limits`.
20. Drag the local variable `ResultString` into the Watch Expression pane as well. Return to the **Steps** tab and click **Step Over** on the toolbar to execute the Property Loader step.

    Notice that the values of the limits in the Watch Expression pane are updated as shown in Figure 6-4.

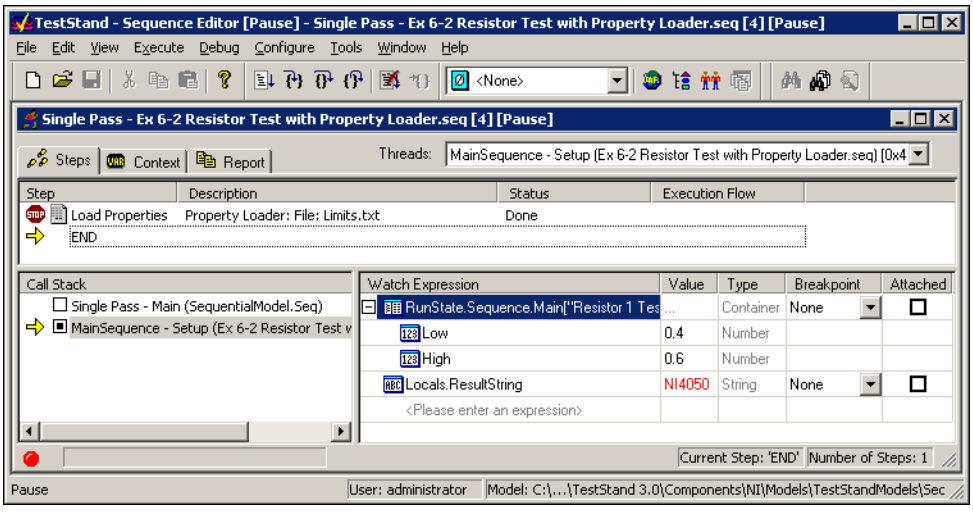


Figure 6-4. Watch Expression Pane

21. Click **Resume** on the toolbar to finish the sequence execution and examine the test report to verify that all limits were updated correctly.
22. Remove the breakpoint that you set on the `Load Properties` step in step 17 of this exercise before continuing to the next exercise.

End of Exercise 6-2

Exercise 6-3 Using Expressions to Load Properties

Objective: To use the Property Loader step type to import different values based on the value of a TestStand expression.

The Property Loader step can be a powerful tool in sequence development. The step allows one sequence file to test many different components, each with a different set of property values. For example, assume that two UUTs need to pass a similar set of tests, except that each UUT has a different set of associated properties. One solution is to create a sequence for each UUT, where each is identical except for the certain properties of their steps. However, it is more practical and efficient to create one sequence and use the Property Loader step to load the correct properties dynamically during the execution of the sequence. This exercise demonstrates how to test for the presence of a particular UUT and, based on the results of the test, load the properties associated with that UUT for the rest of the sequence.

1. Launch Notepad and open the `Limits.txt` text file located in the `C:\Exercises\TestStand I` directory.
2. Change the limits as shown in Figure 6-1.

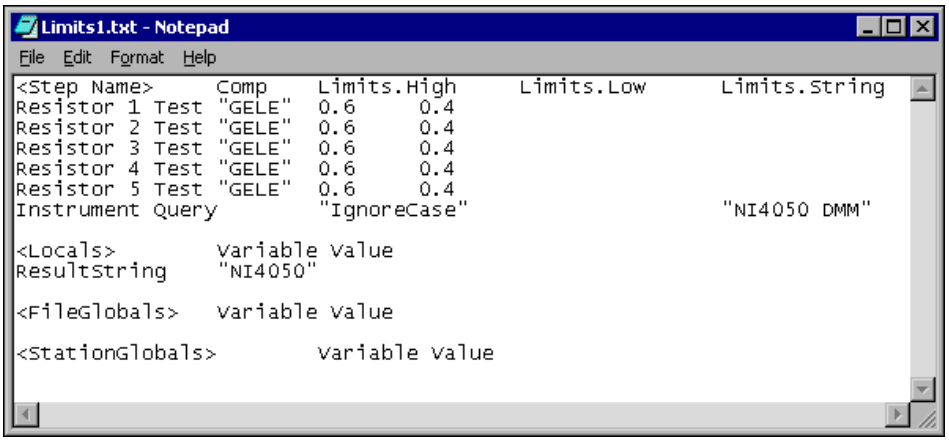


Figure 6-1. Limits1.txt File

3. Select **File»Save As** and save the file as `Limits1.txt` in the `C:\Exercises\TestStand I` directory, but do not close the text file.
4. Change the limits as shown in Figure 6-2.

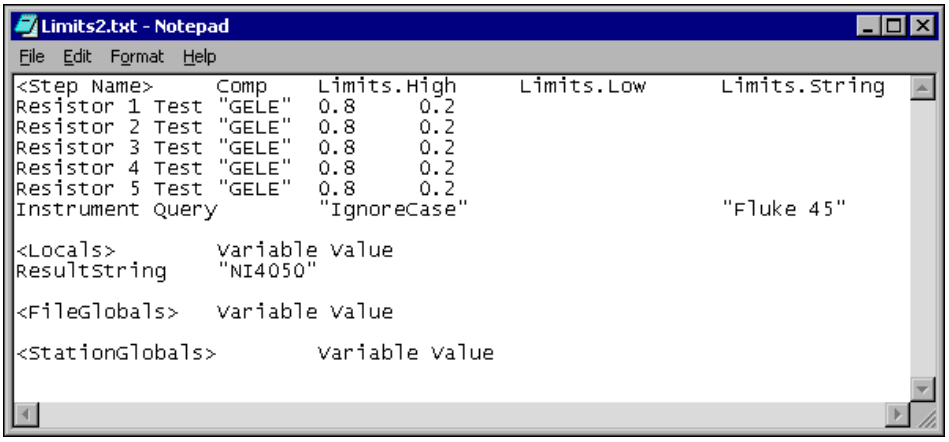


Figure 6-2. Limits2.txt File

5. Select **File»Save As** and save the file as `Limits2.txt` in the `C:\Exercises\TestStand I` directory.
6. Close Notepad.
7. Open the `Ex 6-2 Resistor Test with Property Loader.seq` sequence file located in the `C:\Exercises\TestStand I` directory.
8. Select **File»Save As** and save the file as `Ex 6-3 Resistor Test with Limit Expression.seq` in the `C:\Exercises\TestStand I` directory.
9. From the `MainSequence`, select the **Setup** tab and insert a Message Popup step before the Property Loader step.
10. Name the step `Select Properties to Use`.
11. Right-click the `Select Properties to Use` step and select **Edit Message Settings** from the context menu.
12. Set the fields in the **Text and Buttons** tab of the Configure Message Popup Step dialog box as shown in Figure 6-3.

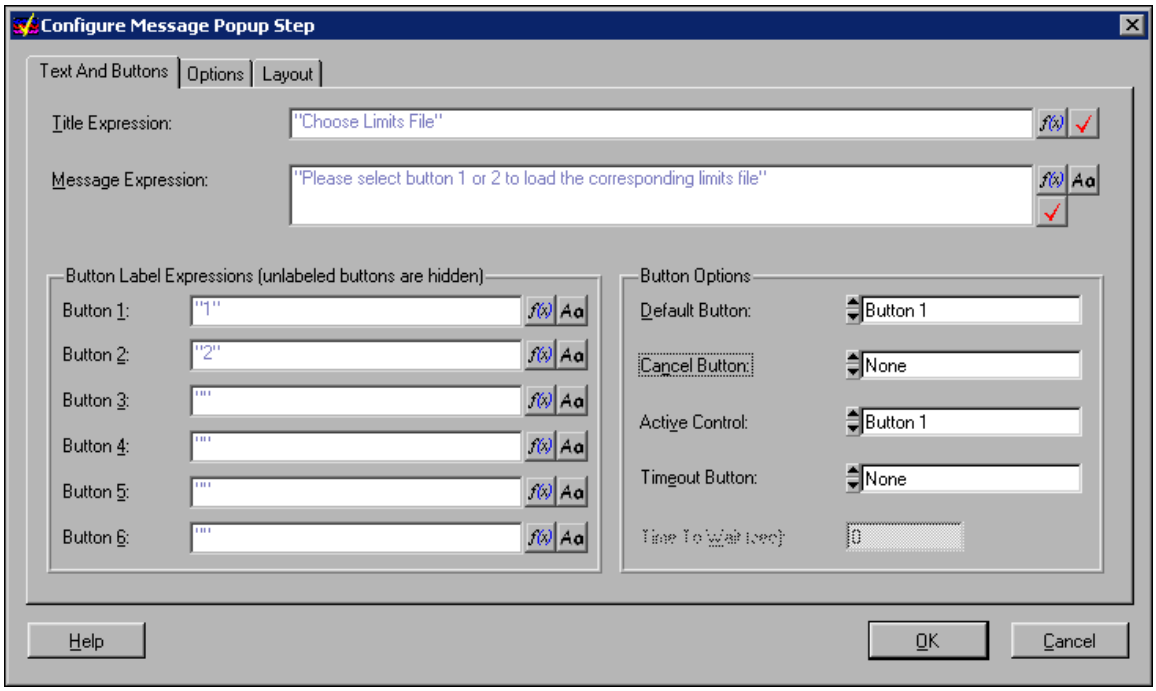


Figure 6-3. Text and Buttons Tab Settings

13. Click **OK** to return to the Sequence File window.
14. Right-click the `Load Properties` step and select **Edit Property Loader** from the context menu.
15. Click the **Source** tab and enable the **Use Expression to Specify File** option.
16. Use the Expression Browser to enter the following text into the **File Name Expression** ring control.

    ```
    "Limits" + Str(RunState.Sequence.Setup["Select Properties to Use"].Result.ButtonHit) + ".txt"
    ```

    This expression loads the file with the name `Limitsx.txt`, where *x* corresponds to the button selected by the operator.

Note The *x* value is obtained from the `Result.ButtonHit` property value, not the expression specified for the button. For example, if Button 1 was labeled `25`, `Result.ButtonHit` would still yield the value `1` and not the value `25`. This exercise also could have enabled the response text box within the Message Popup step and had the operator enter a `1` or a `2` rather than using buttons.

17. Click **OK** to return to the Edit Property Loader dialog box. Figure 6-4 shows the resulting dialog box.

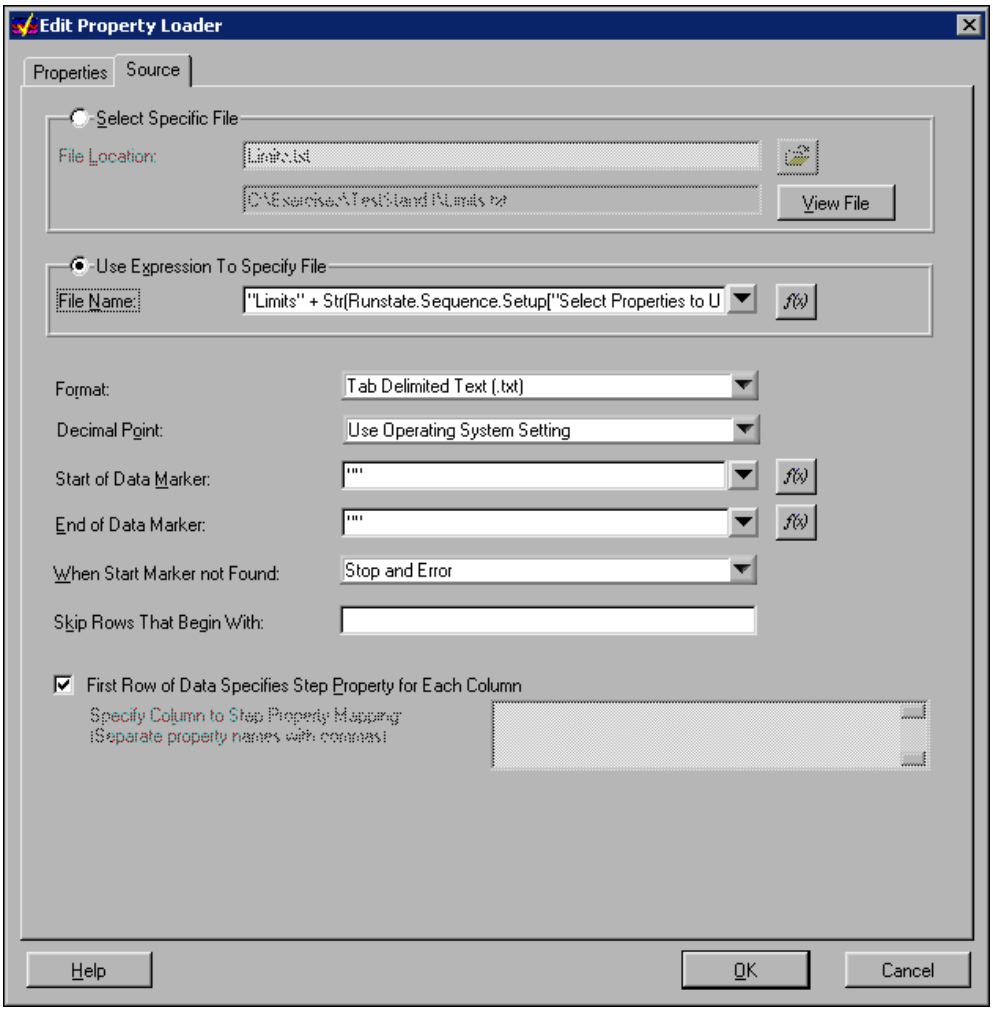


Figure 6-4. Edit Property Loader Dialog Box

18. Click **OK** to return to the Sequence File window.
19. Save the changes to the sequence.
20. Selecting **Execute»Test UUTs**, execute the sequence and follow the steps below.
    a. Select **Button 1** in the Choose Limits File dialog box for the first UUT.
    b. Select **Button 2** when you test the second UUT.
    c. Stop the execution after the second UUT and examine the rest of the report.

    The limits for the first UUT should be from `Limits1.txt` and the limits for the second UUT should be from `Limits2.txt`.
21. Close the Execution and Sequence File windows.

End of Exercise 6-3

Self Review

1. How can you import and export properties used within your sequence to or from a database, file, or clipboard?

2. Give an example of when you would use the Import/Export Properties tool?

3. What is the Property Loader step used for?

Notes

7

Configuring TestStand

Lesson 7: Configuring TestStand

In this lesson, you will:

- **Use the Station Options dialog box to set preferences for your TestStand station**
- **Use the Report Options dialog box to customize the generation of report files**
- **Use the Search Directories dialog box to modify the search paths that TestStand uses to find files**

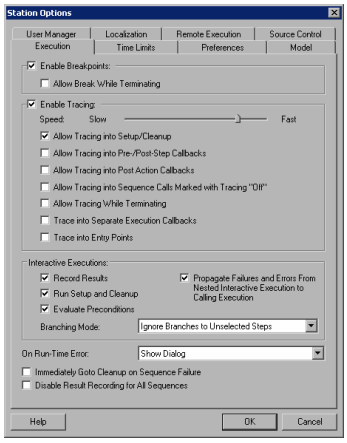
Introduction

You can modify many features of TestStand using configuration dialog boxes. The following three configuration dialog boxes are used most commonly in TestStand:

- **Station Options**—Set preferences for your TestStand station.
- **Report Options**—Customize the generation of report files.
- **Search Directories**—Modify the search paths TestStand uses to find files.

Station Options Dialog Box

- Set preferences for a TestStand station
- Select Configure»Station Options



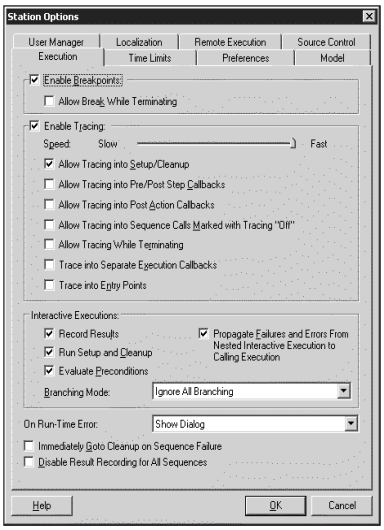
Station Options Dialog Box

Use the Station Options dialog box to set preferences for a TestStand station. The settings affect all the sequence editor and operator interface sessions that run on that computer. Select Configure»Station Options to open the Station Options dialog box. The Station Options dialog box contains the following options:

- **Execution**—Contains operations for breakpoints, tracing, and interactive execution.
- **Time Limits**—Specifies time limits for executions. If you specify a time limit, choose an action to take when the time limit expires.
- **Preferences**—Specifies general options for the TestStand station, such as the name of the test station.
- **Model**—Specifies the process model file for the station as a whole and whether each individual sequence may specify its own process model file.
- **User Manager**—Specifies whether TestStand enforces user privileges. It also specifies the location of the user manager configuration file.
- **Localization**—Specifies the language in which to display text and other region-specific settings.
- **Remote Execution**—Specifies whether remote test stations may run sequences on this test station.
- **Source Control**—Specifies general options that apply to source control operations in TestStand.

Station Options: Execution Tab

Specifies the options for executing sequences

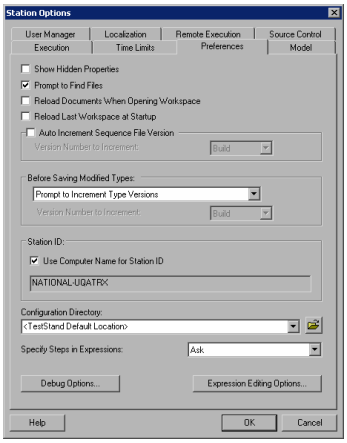


Station Options: Execution Tab

The Execution tab has options for breakpoints, tracing, and interactive execution. Refer to the *TestStand Help* for more information about the options available on the Execution tab of the Station Options dialog box.

Station Options: Preferences Tab

Specifies general options for TestStand

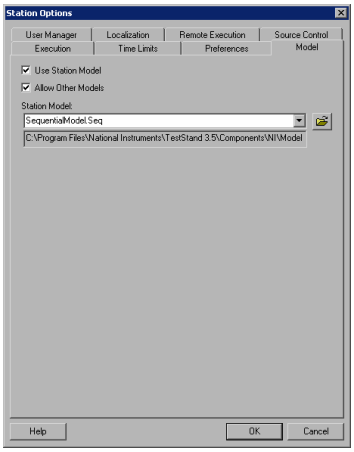


Station Options: Preferences Tab

The Preference tab specifies general options for TestStand. Refer to the *TestStand Help* for information about the options available on the Preferences tab of the Station Options dialog box.

Station Options: Model Tab

Specifies the process model options for the station and sequences



Station Options: Model Tab

The Model tab specifies the process model file for the station as a whole and whether each individual sequence may specify its own process model file. Refer to the *TestStand II: Customization Course Manual* and the *TestStand Reference Manual* for more information about the different process models that ship with TestStand. Refer to the *TestStand Help* for more information about the Model tab of the Station Options dialog box.

Report Options Dialog Box

- **Available options include:**
  - Report format (ASCII-text, HTML, or XML)
  - Insert arrays as tables or graphs
  - Result filtering
    Example: only report steps that fail
  - Numeric formatting
  - Report color schemes (HTML and XML only)
  - Report file pathname(s)
  - On-The-Fly Reporting
- **Select Configure»Report Options**

Report Options Dialog Box

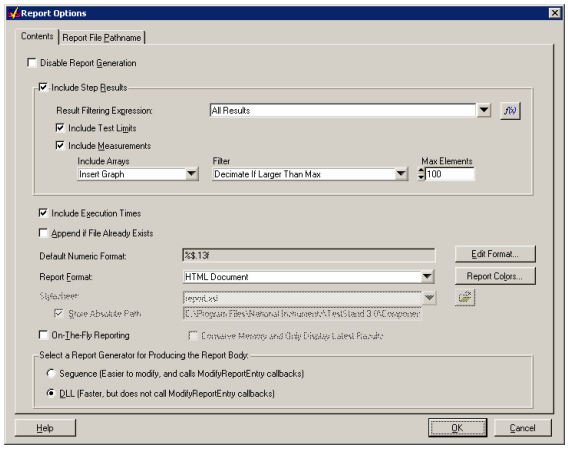
Use the Report Options dialog box to customize the generation of report files. When you use the Test UUTs and Single Pass items in the Execute menu, the settings in this dialog box apply to all sequences that you run on a station. To launch the Report Options dialog box, select **Configure» Report Options**.

When you select the Report Options command, TestStand calls the Config Report Options entry point in the default process model. Thus, while the dialog box is active in the sequence editor, the `Running` tag appears on the left side of the status bar of the Sequence Editor window.

The Report Options dialog box contains the following tabs:

- **Contents**—Specifies general options for report generation, such as the type of data to include in the report, the report format, whether TestStand generates a report after testing a UUT, or whether TestStand generates a report concurrently with the execution.
- **Report File Pathname**—Provides a graphical indicator to illustrate how options you select affect the names and contents of the report files. You can specify that all batch and UUT reports reside in the same file, that all reports reside in separate files, or one of several intermediate configurations.

Report Options: Contents Tab



Report Options: Contents Tab

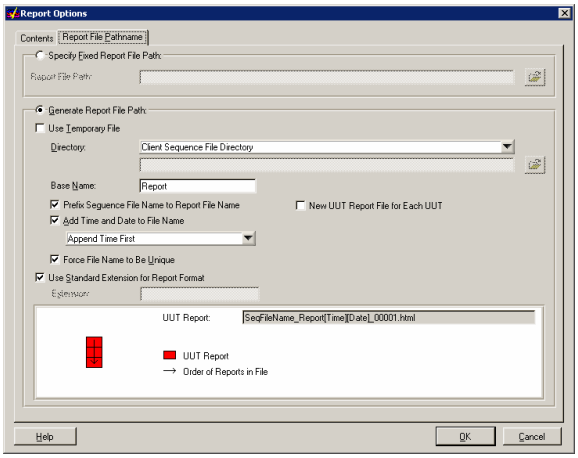
The Contents tab contains options which allow you to configure report properties, such as which result or elements are included in the report, the report format (ASCII text, HTML, and XML), report color (HTML and XML only), which results to include, whether to insert array(s) of data into the report as a table or graph, and the formatting of numeric data.

You can also specify whether to use the sequence method or DLL method to generate the report body. Unless your report body is rather large you will probably not notice a difference in the amount of time it takes for the sequence option or DLL option to generate the report body. These two methods are discussed in greater detail in the *TestStand II: Customization Course Manual*.

By default, the TestStand process model generates the test report after your main sequence completes. The process model generates the report from the execution information that TestStand collects in the result list. You can select the On-The-Fly Reporting option so the process model uses Post Result callbacks to append new formatted results to the report. This is useful in many cases, such as running tests where a reboot or computer crash might occur or you need to view a partial report before testing is complete.

Refer to the *TestStand Help* for more information about the Contents tab of the Report Options dialog box.

Report Options: Report File Pathname Tab



Report Options: Report File Pathname Tab

Use the Report File Pathname tab to configure report properties pertaining to the file pathname that TestStand uses to save the report to disk. You can specify a fixed pathname to use for all report files or you can specify options that the report generator uses to generate report file pathnames. The Report File Pathname tab of the Report Options dialog box contains a Generate Report File Path section. This section provides a graphical indicator to illustrate how the options you select affect the names and contents of the report files.

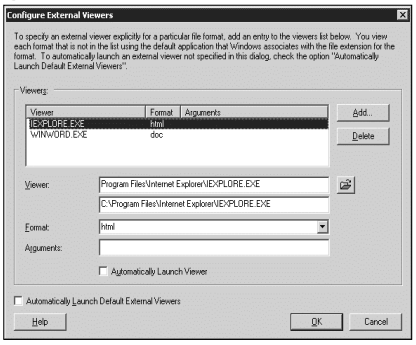
The controls located on the Report File Pathname tab vary according to the process model you use.

- **Specify Fixed Report File Path**—When you enable this option, you enable the Report File Path control in which you can specify a pathname that applies to all report files. You must specify an absolute pathname. Each report file that the report generator creates overwrites the previous report file unless you enable the **Append if File Already Exists** option on the Contents tab.
- **Generate Report File Path**—When you enable this option, you enable controls in which you can specify how the process model builds a pathname for report files. The configurable controls include whether to include the sequence filename, time, date, or UUT serial number in the pathname. You can also specify a base report name to be used in every report file pathname and specify whether to force each filename to be unique.

Refer to the *TestStand Help* for more information about the Report File Pathname tab of the Report Options dialog box.

Configuring External Viewers Dialog Box

- TestStand allows external report viewers such as Word, Netscape, and Internet Explorer
- Select Configure»External Viewers



Configure External Viewers Dialog Box

You might prefer to view the test report in an external application, such as Microsoft Word, Netscape, or Internet Explorer. Select **Configure» External Viewers** to specify the application that TestStand launches to display the report.

Select **View»Launch Report Viewer** in an active Execution window to use the external application to view reports.

To display the results of your UUT, configure this dialog box to launch an external viewer, such as Internet Explorer.

Exercise 7-1: Customizing Report Generation (Optional)

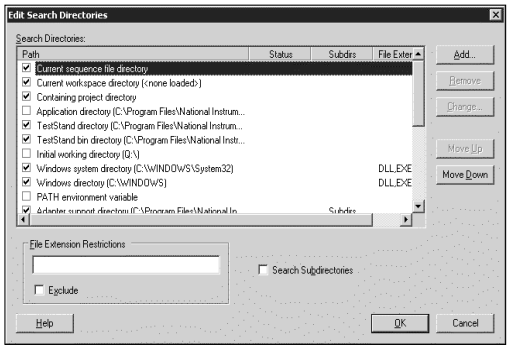
Objective: To demonstrate the options for configuring report generation in TestStand.

Estimated Time: 20 minutes

Refer to page 7-16 for instructions for this exercise.

Edit Search Directories Dialog Box

- Customize the search path for finding files
- Select Configure»Search Directories



Edit Search Directories Dialog Box

Use the Edit Search Directories dialog box to customize the search paths for finding files. The Edit Search Directories dialog box displays a list of paths in which the higher paths take precedence over the lower paths. The list contains a default set of paths. When you enable a path, TestStand includes the path in the overall search path. To reorder paths in the list, select a path and click the **Move Up** or **Move Down** buttons. Click the **Add** button to add a custom directory search path.

When you select a path, the following controls display the settings for the path:

- **File Extension Restrictions**—Searches only for files with specific filename extensions. For example, to search for only DLLs and executable files, enter the following string: `DLL, EXE`.
- **Exclude**—Searches for all files except those files with specific extensions, enable the option. A tilde (~) is visible at the beginning of the File Extension Restrictions column for the row that you have selected when this option is enabled.
- **Search Subdirectories**—Specifies whether to include all subdirectories within the selected path in the overall search path.

Edit Search Directories Dialog Box (Continued)

Edit Search Directories Dialog Box (Continued)

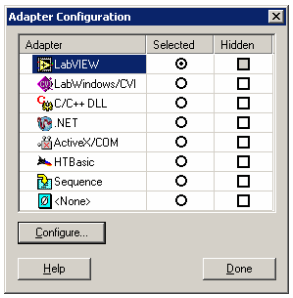
Important Notes on Search Directories

Note If you add a large directory tree to the search paths and specify the **Search Subdirectories** option, TestStand must search the entire tree each time it locates a file. You can improve performance by using the **File Extension Restrictions** control to limit the types of files for which TestStand searches a directory tree. For example, if a directory tree contains only VI files, specify `VI` in the **File Extension Restrictions** control to prevent TestStand from searching the directory tree for files of other types such as `.seq` and `.dll`.

Note For best performance, move the following search paths to the end of the search path list: the search paths you add that refer to large directory trees, directories on network drives, and paths that contain files which you use infrequently.

Adapter Configuration Dialog Box

- Customize which adapters appear in the Adapter Ring Control
- Configure individual adapters
- Select Configure»Adapters



Adapter Configuration Dialog Box

The Adapter Configuration dialog box allows you to change settings that apply to each adapter and to specify the selected adapter for the new steps you create. Select **Configure»Adapters** to launch the Adapter Configuration dialog box.

Click the radio button in the Selected column of the Adapter Configuration dialog box to select an adapter . You can also use the Adapter ring control on the Sequence Editor toolbar to specify a module adapter. The selected adapter only applies to step types that can use any module adapter, such as the Action, Numeric Limit Test, Multiple Numeric Limit Test, String Value Test, and Pass/Fail Test step types. Select an adapter in the Adapter Configuration dialog box and click **Configure** to configure an adapter. The Configure button launches an adapter-specific dialog box for configuring the adapter.

You can also remove an adapter from the ring controls that list adapters by placing a checkmark in the Hidden checkbox in the Hidden column. Remove the checkmark from the Hidden checkbox to display the adapter in the adapter ring controls again.

Lesson 7: Summary

- **Use the Station Options dialog box to set preferences for your TestStand station, including:**
  - Execution
  - Preferences
  - Models
- **Use the Report Options dialog box to configure report attributes, including:**
  - Report format (ASCII-text, HTML, and XML)
  - Report color schemes (HTML and XML only)
- **Use the Edit Search Directories dialog box to customize the search path TestStand uses to find files**
- **Use the Adapter Configuration dialog box to modify the Adapter ring control and individual adapter settings**

Summary

This lesson described how to use the following four dialog boxes to configure TestStand:

- **Station Options**—Sets preferences for your TestStand station. Changes made in this dialog box affect all sequence files that you run on the station, whether you run them in the operator interface or the sequence editor.
- **Report Options**—Configures various attributes of the report that TestStand generates once the UUT completes. You can change such attributes as color, pathnames, and report format.
- **Edit Search Directories**—Modifies the search path TestStand uses to locate files. You can add, delete, and reorder directories from this dialog box.
- **Adapter Configuration**—Modifies the settings for adapters and their respective code modules. You can remove adapters from the adapter ring control and set different ways to execute certain types of code modules.

Exercise 7-1 Customizing Report Generation (Optional)

Objective: To demonstrate the options for configuring report generation in TestStand.

By default, TestStand generates reports after the execution of a test sequence. The report format is determined by the user-defined Report Options, which can vary depending on the process model that is being used. This exercise demonstrates some of the report generation options available in TestStand.

Report Options for the Sequential Process Model

The process model defines how TestStand generates the report. Reporting options change depending on the process model you select. These changes reflect the functionality of each process model. In this exercise, you will examine the report options available when using the Sequential process model.

1. Select **Configure»Report Options** to display the Report Options dialog box, as shown in Figure 7-1.

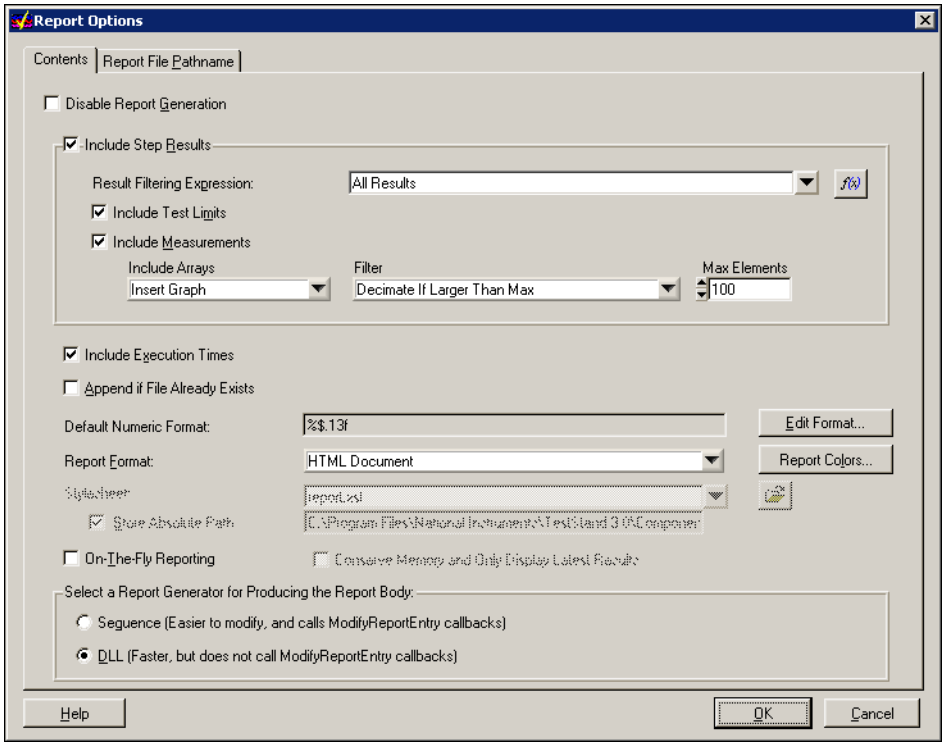


Figure 7-1. Report Options Dialog Box

In the Report Options dialog box, you can select the types of information to add to the test report, including the measured values, test limits, execution times, and array data of each step. You can also format array data as a table or a graph and append information to an existing file.

Note In order to record array data to the report, the array must exist within the `Locals.ResultList` container and the appropriate flags must be set. Refer to the *TestStand II: Customization Course Manual* and the *TestStand Help* for more information about recording array data to a report.

Additional Information

Click **Help** for information about the options on the Report Options dialog box.

2. Click **Edit Format** to launch the Numeric Format dialog box shown in Figure 7-2. Observe the numeric format options.

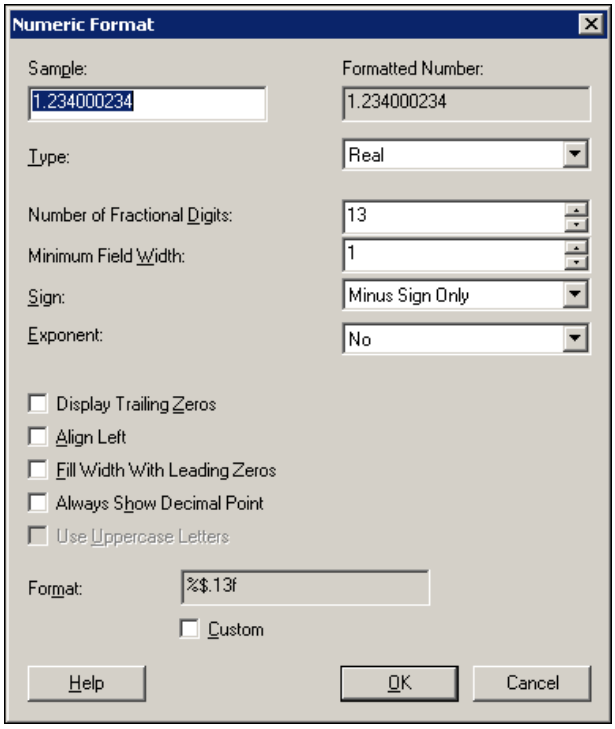


Figure 7-2. Numeric Format Dialog Box

3. Click **OK** to close the **Numeric Format dialog** box.
4. Click the **Report File Pathname** tab. The tab should appear as shown in Figure 7-3.

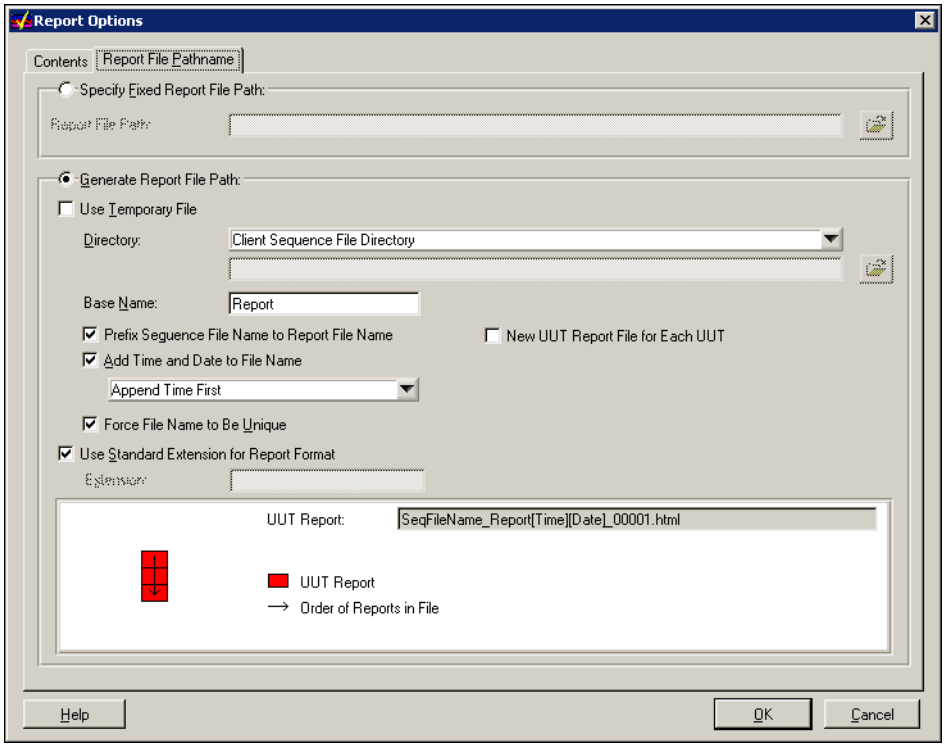


Figure 7-3. Sequential Process Report File Pathname Tab

Additional Information

You can use the Report File Pathname tab to specify a fixed pathname to use for all report files or specify options that the report generator uses to generate report file pathnames. You can generate the report as a temporary file or specify the file to be saved to disk. From this tab, you can also specify the sequence name as a prefix in the filename, add the time and date to the filename, and force a unique filename.

Click **Help** for more information about the options available on the Report File Pathname tab of the Report Options dialog box.

The Report File Pathname tab of the Report Options dialog box contains a Generate Report File Path section. This section provides a graphical indicator to illustrate how the options you select affect the names and contents of the report files.

5. Click **OK** to close the Report Options dialog box.
6. Open the `Computer.seq` sequence file located in the `<TestStand>\Examples\Demo\C` directory. Run the sequence using the Test UUTs or Single Pass entry point and analyze the report.

7. Select **Configure»Report Options** to launch the Report Settings dialog box.
8. Select different options for creating different types of reports. View the reports to see the effects of your changes.

End of Exercise 7-1

Self Review

1. What are station options used for?

2. If you wanted to slow down tracing to trace into your entry points, which option would you configure in the Station Options dialog box?

3. What are the different default report formats available in TestStand?

4. What is the Result Filtering expression used for?

5. To display an array in your report, what option(s) would you configure?

6. The paths listed at the top of the search directory list take precedence over those at the bottom. True or False

7. How would you increase search performance of the TestStand search directories?

Notes

Notes

8

User Management

Lesson 8: User Management

In this lesson, you will:

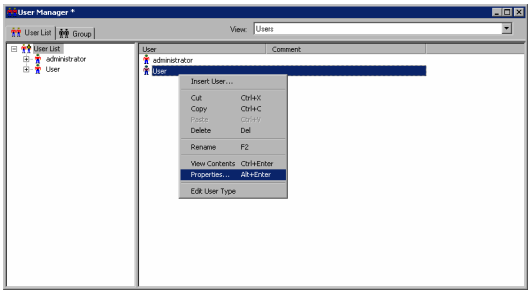
- **Use the User Manager window to add users**
- **Use the User Manager window to customize privileges of users**

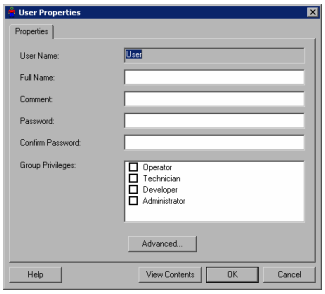
Introduction

In a test environment, several people often interact with the test executive software for a variety of purposes. For example, you might have an operator who runs tests and an engineer who creates the tests. This lesson introduces user management in TestStand. You will learn how to use the User Manager window to add users and customize their privileges.

User Manager Window

- The User Manager window includes predefined user groups
- Select View»User Manager





User Manager Window

Use the User Manager window to view and edit the user list, passwords, and privileges of each user. TestStand limits the functionality of the sequence editor and operator interfaces according to the privilege settings that have been defined in the User Manager window for the current user.

Select **View»User Manager** to open the User Manager window. In the User Manager window, use the View ring control at the top right to access the list of users or to access the list of types that TestStand uses to store user privileges. You can access the following views from the User Manager window:

- **Users View**— Select **Users** from the **View** ring control to access the list of users. You can use this view to add new users or to modify privileges and other properties of existing users. Click the **Users View** tab, right-click the right pane of the User Manager window, and select **Insert User** from the context menu to create a new user. Double-click the user or select **Properties** from the right click menu to access the user properties. Enter user information, such as the user name and password, in the User Properties dialog box that launches. The initial privileges for the new user are determined by the user groups you select. This information is updated in the users.ini file in the <TestStand>\Cfg directory. TestStand maintains a user list to evaluate the user privileges during login. The Users view includes the following tabs:
- **User List**—Contains a list of current users. Each entry contains properties that define that user's login name, login password, and TestStand privileges.

User Manager Window (Continued)

User Manager Window (Continued)

- **Group**—Contains a list of groups you can assign users to. TestStand has several predefined user groups. A group defines a set of values for the properties in the User data type. When you create a new user, you can initialize the values for a new user by assigning that user to one or more groups. If a user is a member of more than one group, the user will have all privileges granted by each of their groups. Users can also be granted individual privileges in addition to their group privileges.

  Only users who have the Configure.EditUsers privilege, such as administrators, can edit the existing groups or create new groups with specific privilege settings. If you make changes to the privileges in a group, your changes affect the privileges for users who are already members of that group unless the modification would take away a privilege that the user has from another group or a privilege that has been explicitly granted to that user.
- **Types View**—Select **Types** from the View ring control to access the list of types that the User Manager uses. You can use this view to add new properties to the User standard data type or to create your own custom data types. Refer to the *TestStand Help* for more information about the Types View of the User Manager window.

Default User Groups

Privilege / Group	Operate	Debug	Develop	Configure Users
Operator	Yes	No	No	No
Technician	Yes	Yes	No	No
Developer	Yes	Yes	Yes	No
Administrator	Yes	Yes	Yes	Yes

Default User Profiles

The following predefined user groups are included with TestStand—`Operator`, `Technician`, `Developer`, and `Administrator`. The high-level privilege groups associated with each group are shown in the table above.

User Manager Demo

1. Add a new group that does not allow the user to run selected steps but gives them all other privileges
2. Add a new user that is configured with the group you just created
3. Login as the new user you created and verify that you cannot run selected steps

User Manager Demo

Refer to Exercise 8-1 (Optional) on page 8-9 for instructions for using the User Manager window to modify user privileges.

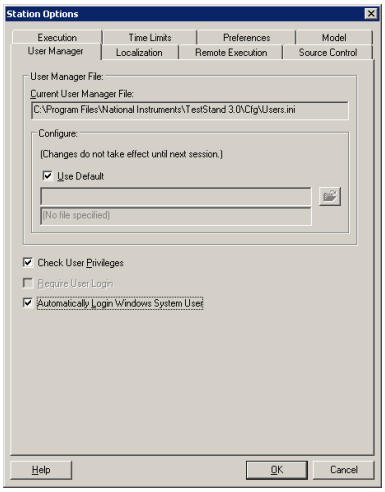
Exercise 8-1: Setting User Login Access and Privileges (Optional)

Objective: To set user login and access privileges in the user manager and create a new group.

Estimated Time: 15 minutes

Refer to page 8-9 for instructions for this exercise.

Windows System User



- TestStand can automatically login the Windows system user
- Select the Automatically Login Windows System User option on the User Manager tab of the Station Options dialog box

Windows System Users

You can configure TestStand to automatically login the current Windows system user. This option is configured on the User Manager tab of the Station Options dialog box. If this option is selected and there exists a user configured with the same user name as the current Windows system user, TestStand logs this user into the sequence editor automatically. Otherwise, TestStand displays the Login dialog box.

For additional login examples, search using *login* as the search term on National Instruments Developer Zone, `ni.com/zone`.

Note You must configure users in the TestStand User Manager if you want TestStand-specific data and privileges to be associated with the user.

Lesson 8 Summary: User Management

- **Create multiple users with different capabilities**
- **Create a variety of groups using different combinations of privileges**
- **Limit modification of information in the user manager to users with the Configure.EditUsers privilege**
- **Configure TestStand to automatically login the Windows system user**

Summary

A production environment involves a wide variety of users. For each type of user, TestStand allows you to provide access-level privileges. You can manage users through the user manager.

The `Users.ini` file located in the `<TestStand>\Cfg\` directory contains the user list. This is the same directory in which the file `StationGlobals.ini` maintains persistent information about station global variables.

Note When deploying your test system, you should add the contents of the `<TestStand>\Cfg` directory to the deployed workspace file in order to reproduce a test station.

Exercise 8-1 Setting User Login Access and Privileges (Optional)

Objective: To set user login and access privileges in the user manager and create a new profile.

In TestStand, each user has a set of associated privileges. Privileges restrict the functionality available to each user in the sequence editor and the operator interface. You can use the user manager to add new users and groups and to set user privileges.

Note You can also change the definition of the user type to create new privilege properties.

Part A: Creating a New User and Using the User Manager

1. Select **File»Login** to set the current user to `Administrator`.

   Notice that the status bar at the bottom of the Sequence File window displays the user currently logged in.

2. Click **User Manager** on the toolbar, shown at left, or select **View»User Manager**. Figure 8-1 shows the User Manager window.

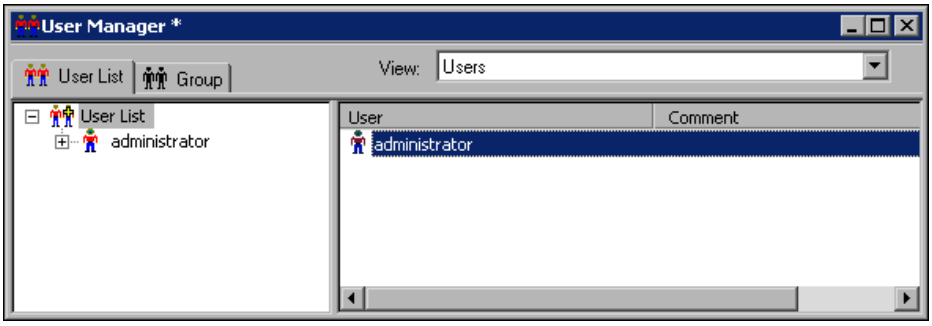


Figure 8-1. User Manager Window

3. Insert a new user.
   a. Right-click the right pane of the User Manager window and select **Insert User** from the context menu.
   b. Name the user `Joe`.
   c. Right-click the new user and select **Properties** from the context menu.

d. Complete the text fields in the user properties dialog box.

Full Name: `Joe Tester`

Comment: `Just an average tester`

Password: `Joe`

Confirm Password: `Joe`

The password in this exercise is the same as the user name, but you can enter any password in this field.

e. Check **Operator** in the **Group Privileges** list.

The user group assigns a set of default privileges. The built-in user groups are `Administrator`, `Developer`, `Operator`, and `Technician`.

Figure 8-2 shows the completed user properties dialog box.

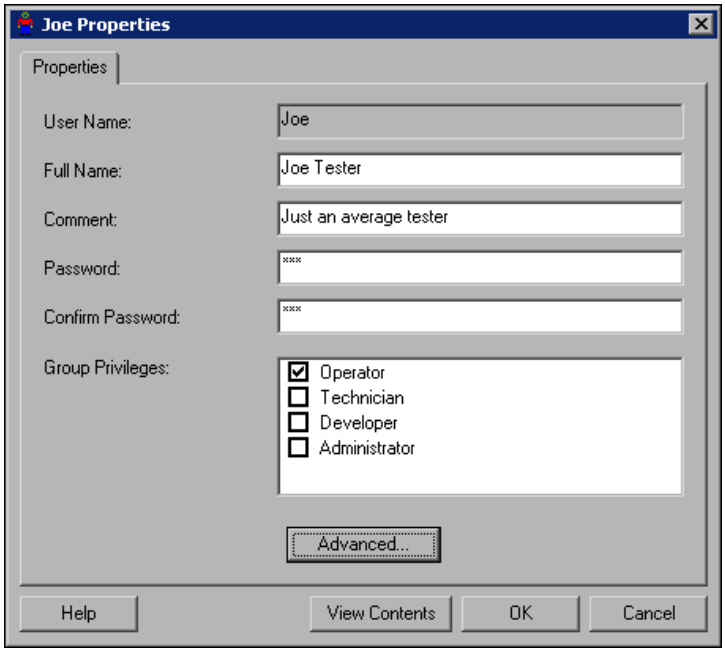


Figure 8-2. New User Dialog Box

f. Click **OK** to exit the user properties dialog box.

4. Observe the privilege levels for `Joe` and `administrator` by clicking the **+** sign next to each user in the left pane and browsing through the privileges trees.

Figure 8-3 shows the privilege values for `Joe`. Notice that `Joe` has different privilege values than `administrator`.

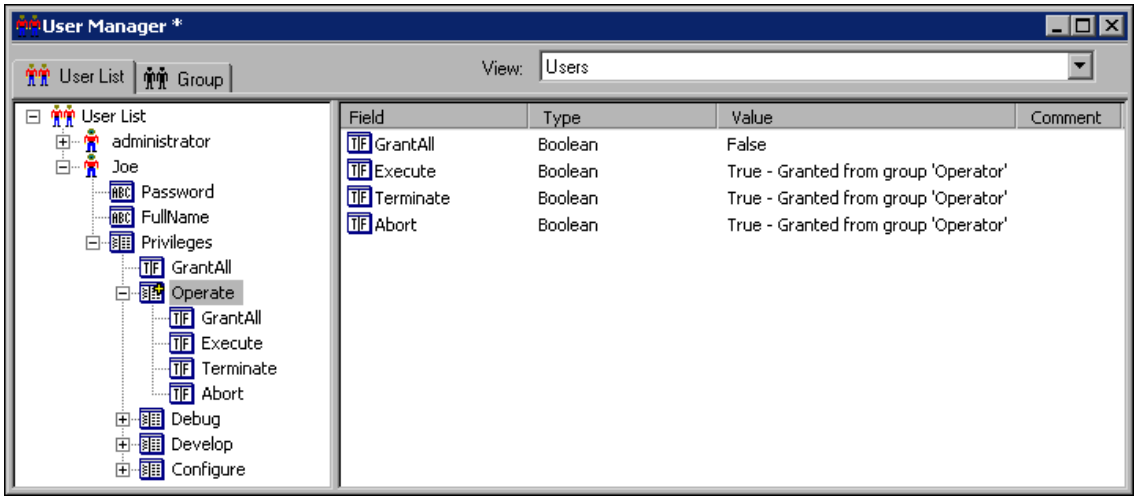


Figure 8-3. Privilege Tree in User Manager

5. To configure privileges individually, double-click a privilege in the right pane or right-click the value in the left pane and select **Properties** from the context menu.

   When you change a value for a user, the change affects only that specific user.

   You can grant individual users privileges in addition to any privileges they are given by their group(s). You cannot, however, take away a privilege which has been given to the user by their group unless you remove the user from that group.

Note Each tree branch contains a node labeled GrantAll **at each level**. If the value of the GrantAll node is set to `True`, all subproperties of that branch are considered to be set to `True` regardless of their individual settings. When the GrantAll node is set to `False`, the individual subproperty values are used. The behavior of the GrantAll node allows you to set groups of privileges by setting a single value to `True`.

6. Select **File»Login** and notice that the User Name ring control contains both `administrator` and the user name `Joe`, created in this exercise, as shown in Figure 8-4.

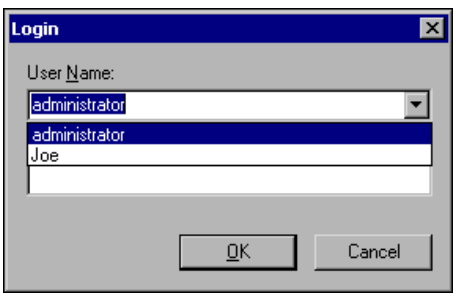


Figure 8-4. Login Dialog Box

7. Before finishing the exercise, be sure to log in as `administrator` so you will have sufficient privileges to complete subsequent exercises.

End of Part A

Part B: Creating a New User Group

In TestStand, you can create your own user groups. TestStand has four default groups; however, it might be desirable to create new groups. Part B demonstrates how to construct a new group and explains the difference between a group and a user.

1. Click the **Group** tab to examine the list of groups.

   The **Groups** tab contains the list of all currently defined user groups. TestStand has four default groups: `Operator`, `Technician`, `Developer`, and `Administrator`.

2. Click the **+** signs next to **Operator** and then **Privileges**. Examine the default privileges assigned to an operator. Figure 8-5 shows the privileges.

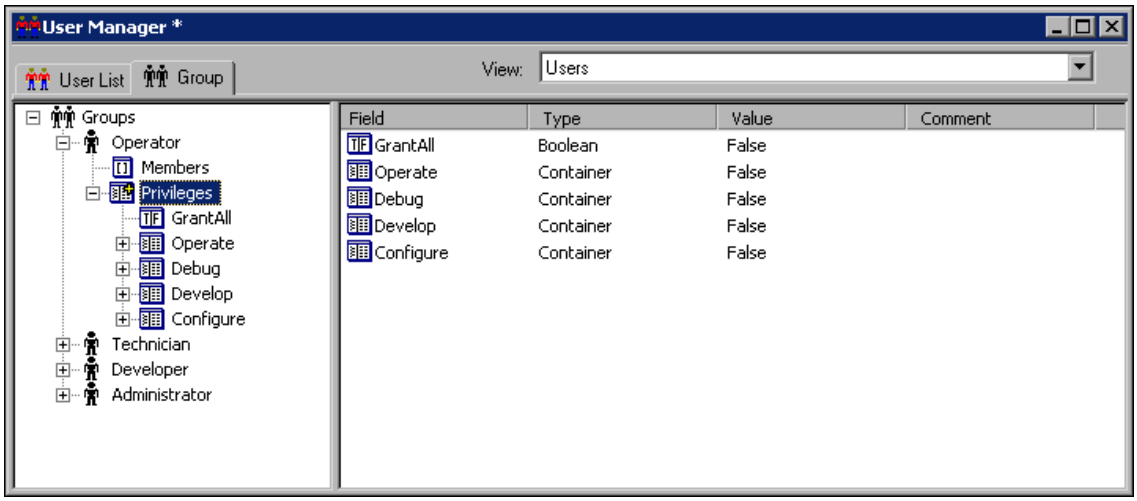


Figure 8-5. Profile Privilege Tree

Examine the privileges for the rest of the groups. You can individually configure all the default groups in the tree.

3. To change the privileges, double-click a privilege in the right pane or right-click a privilege in the left pane and select **Properties** from the context menu.

   Additional Information

   When you change a profile property, the change affects all users who are a member of that group. You should create a new user group when you need to create more than one user with the same privileges because it is easier to assign each user to a group with privileges already configured

than create each user and then set the privileges individually. Now in this exercise you will create a new group called LeadDeveloper and then create a user who is a member of this group.

4. Insert a new user group.
   a. Select **Groups** in the left pane.
   b. Right-click the right pane and select **Insert Group** from the context menu to create a new group.
   c. Name the group `LeadDeveloper`. Figure 8-6 shows the resulting User Manager window.

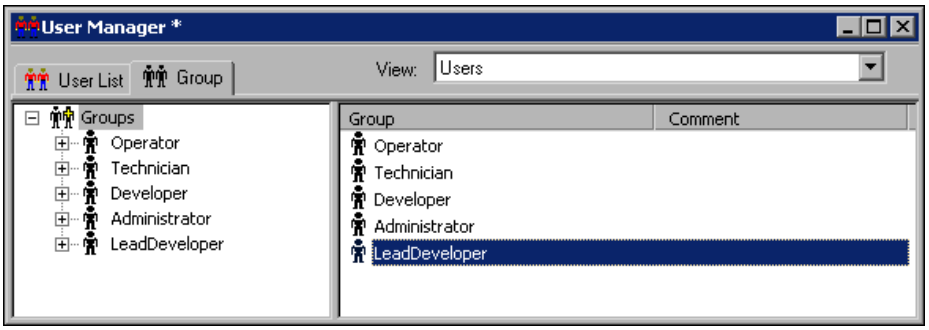


Figure 8-6. New Profile in User Manager

   d. Open the privileges for the `LeadDeveloper` user group you created by clicking the **+** sign next to it in the left pane and then selecting **Privileges**.

      Additional Information
      By default, TestStand creates a new group with the same privileges as the `administrator` group. One difference between administrators and operators is that operators do not have full configure options. You will give the `LeadDeveloper` a few more configure options than an operator, but not as many as the `administrator`.

5. Set the `GrantAll` property value to `False`.
   a. Select **Privileges»GrantAll** from the privilege tree.
   b. Right-click **GrantAll** and select **Properties** from the context menu.
   c. Select **False** in the **Value** ring control and click **OK** to return to the User Manager window.

Figure 8-7 shows the GrantAll Properties dialog box.

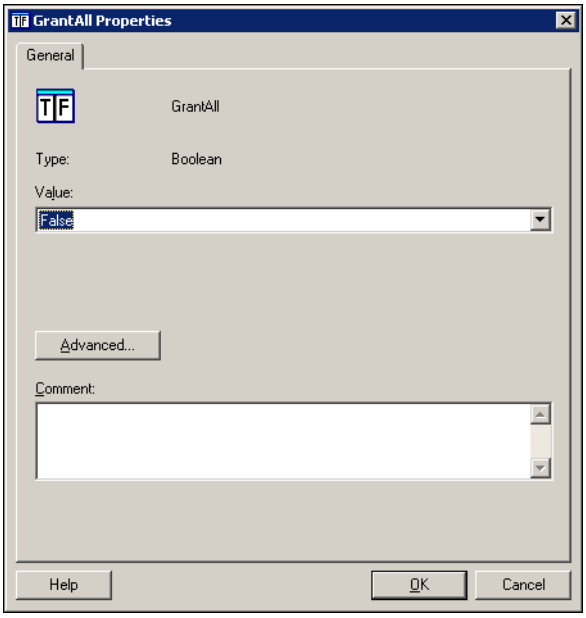


Figure 8-7. Grant All Properties Dialog Box

6. Repeat step 5 for the **Configure»GrantAll** node. This allows you to configure the subproperties individually.

   The LeadDeveloper should be able to do everything except edit users. Because you are configuring privileges individually, the only property you need to change is EditUsers.

7. Double-click `EditUsers` in the right pane and change the Boolean value to `False`.

8. Click **OK** to return to the User Manager window. Figure 8-8 shows the resulting window.

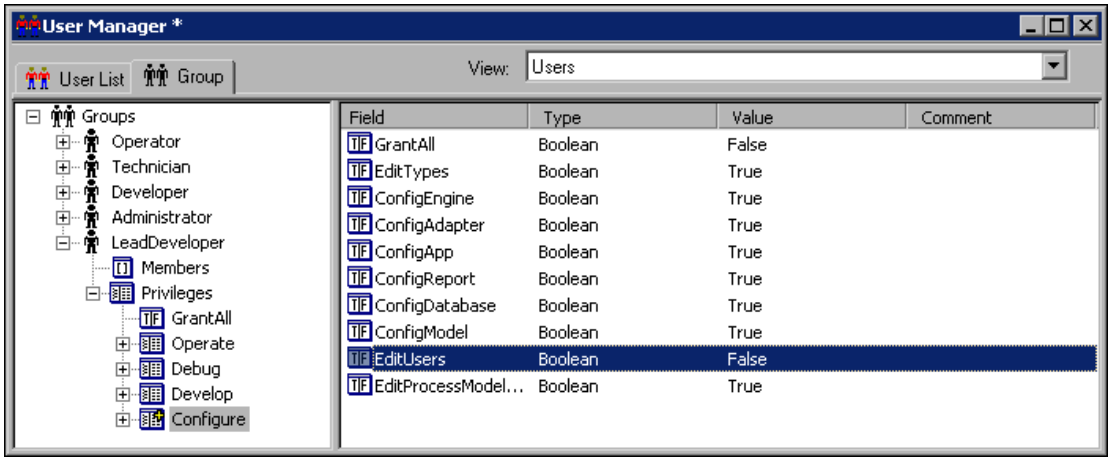


Figure 8-8. Editing a User Value

9. Click the **User List** tab and create a new user.
   a. Right-click the right pane of the User Manager window and select **Insert User** from the context menu.
   b. Name the user `Jane`.
   c. Right-click the new user and select **Properties** from the context menu.
   d. Complete the text fields in the user properties dialog box.

      Full Name: `Jane Lead`
      Comment: `Lead Developer`
      Password: `Jane`
      Confirm Password: `Jane`

   e. Check **LeadDeveloper** in the **Group Privileges** list.

   Figure 8-9 shows the resulting user properties dialog box.

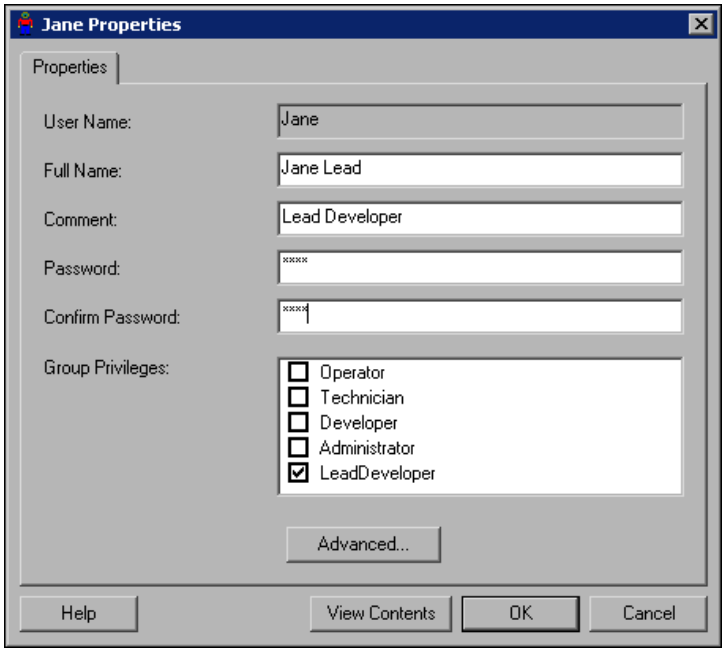


Figure 8-9. New User Dialog Box

Examine the properties for this new user and notice that the only property not set to `True` is the `EditUser` property. Explore the other properties and profiles to become familiar with them.

10. Select **File»Login** and login as `Jane`. Verify that you can operate TestStand normally except for the ability to create or edit users.
11. Close the User Manager window.
12. Select **File»Login** and login as `administrator` before continuing to the next lesson.

End of Part B

End of Exercise 8-1

Self Review

1. List the default user groups in order of their privileges, starting with the user group that has the most privileges.

2. How do you create a new user?

3. Once a user is created, if you make changes to a group that user belongs to, will your changes affect the user's privileges?

4. What is the purpose of the GrantAll property, and what is the effect of setting its value to `True`?

5. How can you have TestStand automatically attempt to login the Windows system user? What happens if the Windows system user is not configured in TestStand?

Notes

9

TestStand Types

Lesson 9: TestStand Types

In this lesson, you will:

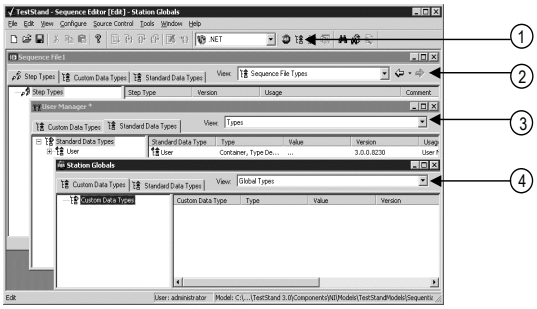
- **Identify different TestStand types**
- **Determine how these types are stored and used**
- **Create custom types**

Introduction

This lesson describes the various TestStand types, how to use them, and how to create and use your own custom types.

What are TestStand Types?

- **Types are defined data structures with two categories:**
  - Step types
  - Data types
- **Each window in the sequence editor has a Types View**



1. Type Palette Button
2. Sequence File Types View
3. User Manager Types View
4. Station Globals Types View

What are TestStand Types?

TestStand types are data structures used to define variables and properties. An example of a custom data type (in this case, a container) is a rectangle that has two numeric subproperties (width and height) and one string subproperty (color). TestStand has two categories of types.

Step Types

Just as each variable or property has a data type, each step has a step type. A step type can contain any number of custom properties. Each step of that type includes the custom step properties in addition to the built-in step properties. While all steps of the same type have the same properties, the values of those properties can differ. The step type specifies the initial values of all the step properties.

Data Types

Data types define station global variables, sequence file global variables, sequence local variables, and properties of steps and step types. You can create and modify your own data types in TestStand, as well as modify the standard TestStand data types.

Grouping TestStand Data Types

Each category is divided into two groups:

- **Standard types**
  - Many built-in types in TestStand environment
- **Custom types**
  - New user-defined types
  - Modified built-in types

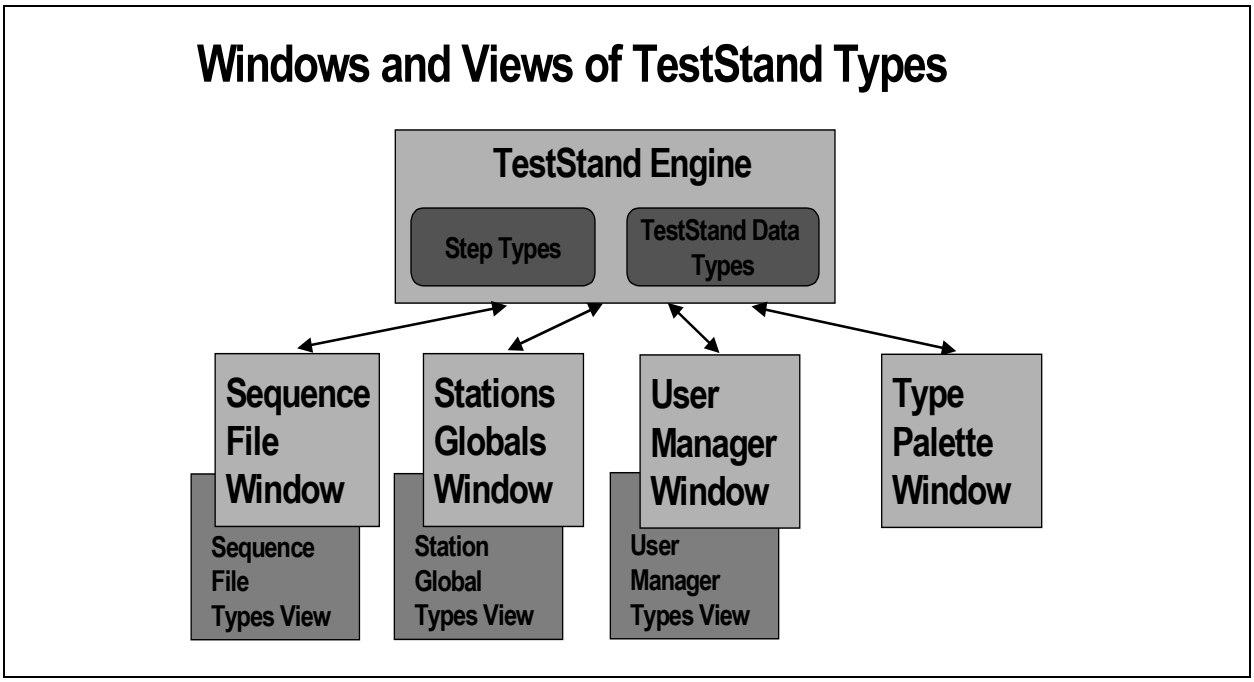
Grouping TestStand Data Types

Data types in TestStand are arranged in two groups: standard and custom data types. Although there are differences between the groups, TestStand treats them similarly.

TestStand defines standard named data types, such as Path, Error, and CommonResults. You can add subproperties to the standard data types, but you cannot delete any of their built-in subproperties. The standard data types are always used in sequence execution.

Custom data types are defined by the developer. You can add or delete subproperties in each custom data type without restriction.

Both standard and custom data types share some common features. One such feature is that although each variable or property you create has the same data structure, the values they contain can be unique to each instance.



Windows and Views of TestStand Types

The TestStand Sequence Editor contains four windows and three views in which you can create, modify, or examine data types and step types. The Sequence File, Station Globals, and User Manager windows each have a corresponding Types view that displays the types corresponding to the file you have open. You also can use the Type Palette window to view and edit Type Palette files. You use the Type Palette files to store the data types and step types that you want to be available in the sequence editor at all times.

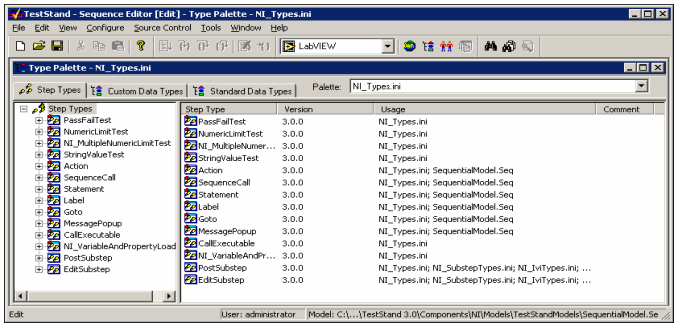
For each type that a TestStand file uses, TestStand stores the definition of the type in the file. Thus, if a custom type is used within a sequence, you can open and use the sequence on another computer. If types were not stored in the file, the target machine would have no definition for custom data types used within a sequence created on another computer.

You can distribute step types and data types you create to other machines by installing your type palette file to the `<TestStand>\Cfg\TypePalettes` directory. You must prefix the file names of the type palettes you install with `Install_`. At startup, TestStand searches the `TypePalettes` directory for type palette files with the `Install_` prefix.

Refer to Chapter 11, *Type Concepts,* of the *TestStand Reference Manual* for more information about concepts that apply to step types, custom named data types, and standard named data types.

Creating Custom Types

TestStand allows you to create your own custom types



Creating Custom Types

One method of customizing the TestStand development environment is to create custom types. In this way, you can define the type of data used in test sequences. You can create, modify, or view data types and step types in the Sequence File, Station Globals, and User Manager windows, or you can use the Type Palette window. Use the Type Palette files to store the data types and step types that you want to be available in the sequence editor at all times.

The Type Palette window contains tabs for Step Types, Custom Data Types, and Standard Data Types.

When you create a new type in the Sequence File Types view of a Sequence File window, the type does not appear in the Insert Local, Insert Global, Insert Parameter, Insert Field, and Insert Step submenus in other Sequence File windows. To use the type in other sequence files, you can manually copy or drag the new type from one Sequence File window to another. A better approach is to copy or drag the new type to the type Type Palette window or to recreate it there. Each type in a Type Palette file appears in the appropriate Insert submenu for all windows. When you save the contents of the Type Palette window, TestStand writes the definitions of the types to the corresponding `*Types.ini` file(s) in the `<TestStand>\Cfg\TypePalettes` directory.

Creating Custom Step Types

- **Custom step types are the most commonly used custom type**
- **By creating custom step types, you control the following:**
  - Properties and values associated with the step
  - Editing the dialog box for the step
  - Run-time behavior of the step
  - Code templates used with the step

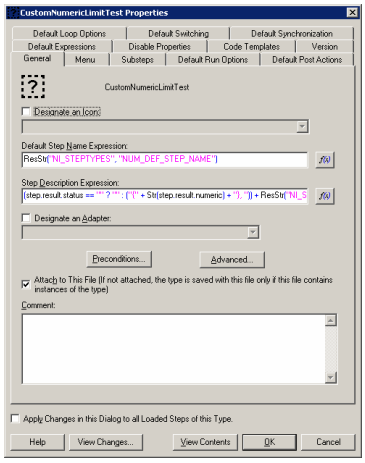
Creating Custom Step Types

You can create and use custom step types when you need to create tests with specific properties not contained within the built-in step types. For example, you might want a test to return an array of data instead of a single Boolean value, like the Pass/Fail Test. When you create custom step types, you can preconfigure all properties for the step. You can develop custom step types that are arbitrarily complex, keep track of large data objects, and perform nontrivial comparisons.

To change or enhance a TestStand built-in step type, copy the files to a User subdirectory, then copy and rename the built-in step type and its supporting modules, and make the changes to the new type and its files. This practice ensures that a newer installation of TestStand does not overwrite your customization and makes it easier to distribute your customization to other users.

Right-click the background of the list view and select **Insert Step Type** from the context menu, to insert a new step type in the Type Palette window or the Sequence Files Types view. Use the Copy and Paste items from the context menu to copy an existing step type.

Configuring Custom Step Types



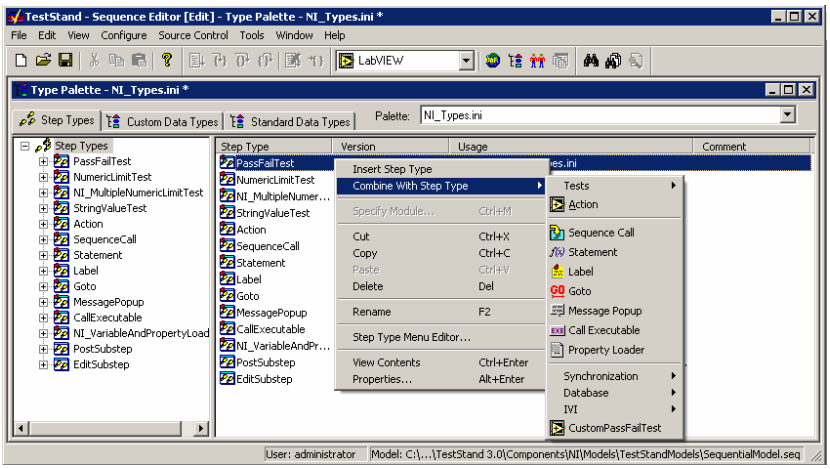
Configuring a Custom Step Type

Every step has the same set of built-in properties. Once you create the custom step type, you can use it as if it were a built-in step type. To propagate the custom step types to other test stations, use the TestStand Deployment Utility to package the necessary files from the `<TestStand>\Cfg` directory.

Configure the properties of a custom step type using the options of the Step Properties dialog box. When you create a custom step type, you can define the default value of each property for the steps that are instances of your custom step type. The following items can also be configured in the Step Properties dialog box.

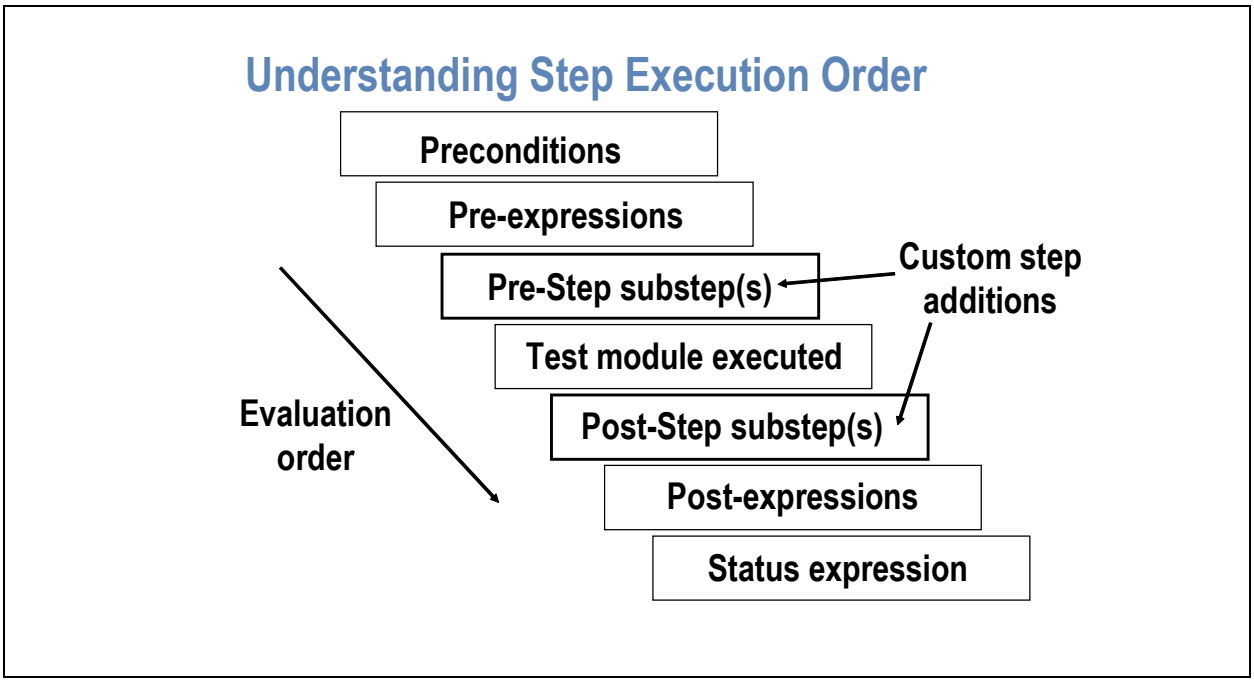
- Specify the name for your step in the Insert Step menu
- Configure pre- and post-step run-time options
- Specify a custom editing dialog box
- Specify code templates
- Disable specific properties

Combining Step Types



Combining Step Types

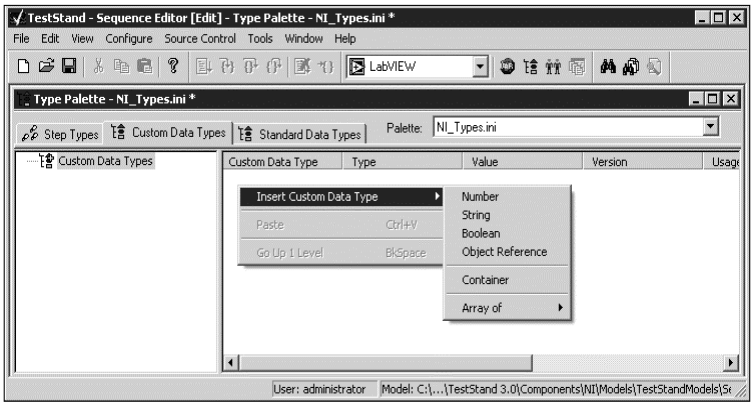
TestStand can combine two step types together to form one new step type that contains the features of both step types. The step you select to right-click and display the Combine With Step Type context menu option determines which step type takes precedence in the event. Some step properties cannot be combined and the properties from one step type are taken priority over the properties of the other step type. TestStand also displays a dialog box describing which properties from each step type were used to create the new combined step type.



Understanding Step Execution Order

When you define pre- or post-step actions for a step type, they execute at the points shown in the slide above. For a more detailed description of step execution, refer to Chapter 3, *Executions,* of the *TestStand Reference Manual*.

Creating Custom Data Types



Creating Custom Data Types

To create a custom data type, right-click in the right pane of the **Custom Data Types** tab of the Type Palette window and select **Insert Custom Data Type** from the context menu. Select the data type you want from the context menu, such as number, string, Boolean, object reference, container (clusters in LabVIEW, structures in C-based languages), or Array of these data types. To view values inside these data types, right-click the data type and select **View Contents**.

For example, you might need a custom global or station global data type to keep track of UUT information pertaining to phones. Create a global type named Phone that contains fields to store data such as the serial number, part number, transmission frequency, and the enabled status of various subsystems. Then use variables or properties of type Phone in your sequences and steps to track the information.

Note You also can define data types in the Sequence File Types view of sequence files.

Exercise 9-1: Creating a Custom Step Type

Objective: To create a new Waveform Limit Test step type that performs a limit test on a waveform.

Estimated Time: 20 minutes

Refer to page 9-14 for instructions for this exercise.

Exercise 9-2: Using a Custom Step Type

Objective: To use the Waveform Limit Test step created in Exercise 9-1 in a test sequence.

Estimated Time: 20 minutes

Refer to page 9-24 for instructions for this exercise.

Lesson 9: Summary

- **TestStand has two categories of types**
  - Data types
  - Step types
- **Categories are divided into two groups:**
  - Standard types (predefined)
  - Custom types (user-defined)
- **Each window has a corresponding types view displaying the types used by the variables and steps**

Summary

TestStand types are either standard types, which are pre-defined in TestStand, or custom types, which are user-defined. To view and create the types used by the variables and steps in a sequence file, TestStand provides a Types view in each Sequence File, Station Globals, and User Manager window. You also can use the Type Palette window to view and edit the types that are always loaded in the sequence editor for you to use when you edit sequence files.

Exercise 9-1 Creating a Custom Step Type

Objective: To create a new Waveform Limit Test step type that performs a limit test on a waveform.

In this exercise, you will create a Waveform Limit Test step type that performs a limit test on an array of numeric data. You want to verify that the entire waveform, represented in a numeric array, lies within a high and low limit. If the waveform exceeds these limits, the step status will indicate a failure. Figure 9-1 displays the waveform.

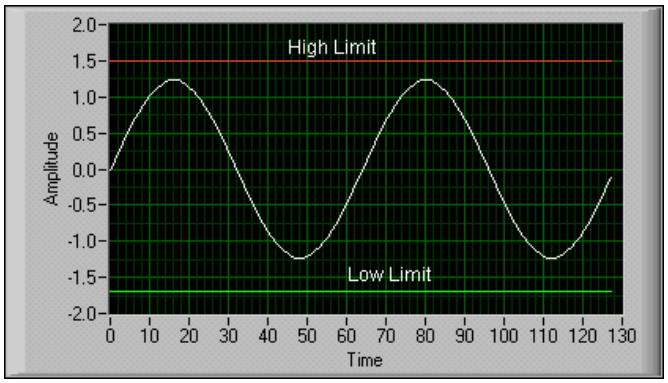


Figure 9-1. Waveform Limit Test Illustration

In order to create such a step type, you will use the existing Numeric Limit Test step type and modify it to add the functionality of testing a waveform against the user-defined limits.

Part A: Modifying the Step Properties

1. Close all open windows within the sequence editor.
2. Click **Type Palette**, shown at left, on the toolbar to launch the Type Palette window.
3. Select `NI_Types.ini` from the **Palette** ring control in the top right corner of the window. Figure 9-2 shows the resulting Type Palette window.

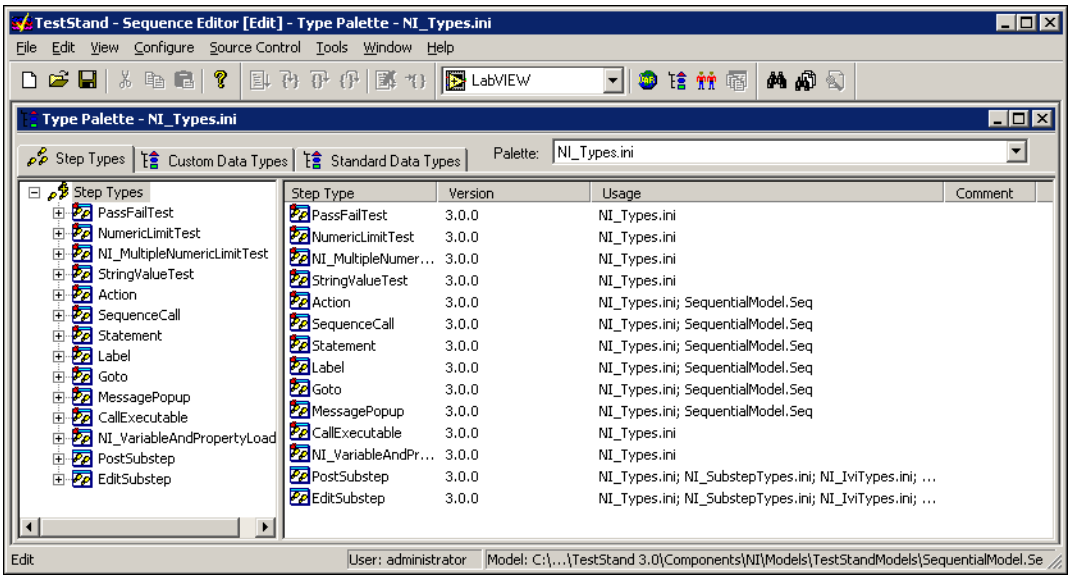


Figure 9-2. NI_Types.ini View in Type Palette Window

The Type Palette window shows all the default step types that are installed by TestStand. You will use the existing Numeric Limit Test step type to create a Waveform Limit Test custom step type.

4. Right-click the `NumericLimitTest` step type in the right pane and select **Copy** from the context menu.
5. Select `MyTypes.ini` from the Palette ring control. Right-click in the right pane and select **Paste** from the context menu. A dialog box opens to warn you that the file you selected references another type. Click **OK** to close the dialog box.

   NumericLimitTest and EditSubstep should appear in the MyTypes.ini palette. The NumericLimitTest step type still references the NumericLimitTest of `NITypes.ini`. You need a unique copy of the step type. Right-click the `NumericLimitTest` property and select **Copy** from the context menu. Right-click in the same pane and select **Paste** from the context menu. `NumericLimitTest_Copy` should appear.

   Additional Information

   The `MyTypes.ini` view is reserved for types that are customized or created by the user. Because you are customizing an existing step type, the `MyTypes.ini` view should show the NumericLimitTest_Copy step type and the EditSubstep step type.

   TestStand copied two step types because the NumericLimitTest step type uses the EditSubstep step type. This is possible because TestStand allows for merging multiple step types to create a single step type with merged functionality.

6. Highlight the `NumericLimitTest_Copy` step type in the right pane. Rename the step `WaveformLimitTest`.
7. Right- click `NumericLimitTest` in the right pane and select **Delete** to delete the extra `NumericLimitTest` step in MyTypes.ini.
8. Click the **+** to the left of the `WaveformLimitTest` step in the left pane to expand the tree view. Figure 9-3 shows the resulting Type Palette window.

   Additional Information
   Notice the properties associated with this step type. You can access these properties using a `Step.`*`xxx`* property path where *`xxx`* is the name of the actual step property. For example, to access the comparison type or `Comp` property of this step, use `Step.Comp` as the property path or lookup string.

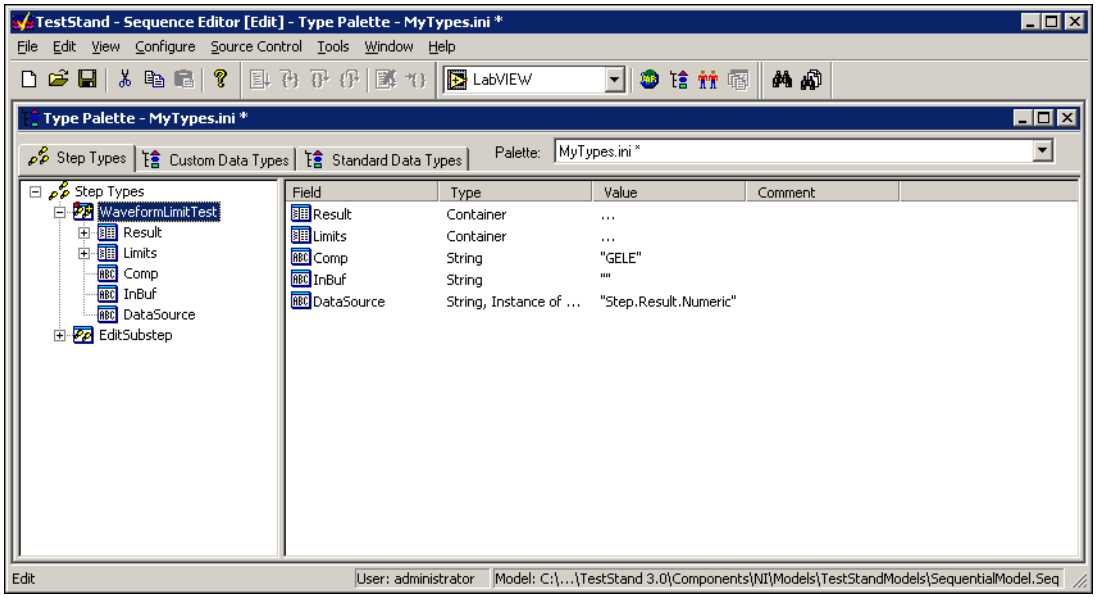


Figure 9-3. Tree View of My_Types.ini

9. Expand the `Limits` property for the `WaveformLimitTest` step and select the `Limits` property in the left pane. In the right pane, notice the `Low` and `High` subproperties. Double-click each subproperty to set the default values. Set the `Low` limit to `-5` and the `High` limit to `5`.

   Additional Information
   Recall that you can access these properties using the property path `Step.Limits.High` and `Step.Limits.Low` in any of the expression windows within the sequence editor.
10. Because the default `NumericLimitTest` step type only returns a single numeric value, you need to remove the `Step.Result.Numeric` property and replace it with a property capable of storing an array of numeric values.

a. Expand the `Result` container in the left pane for the `WaveformLimitTest`.

b. Highlight the `Result` property in the left pane so that all the subproperties appear in the right pane.

c. Highlight the `Numeric` subproperty in the right pane and press the **<Delete>** key to remove this property.

d. Right-click the right pane and select **Insert Field»Array of» Number** from the context menu. The Array Bounds dialog box launches. Enable the **Initially Empty** option so the array is empty and you do not have to set the upper bounds.

Figure 9-4 shows the **Array Bounds** dialog box.

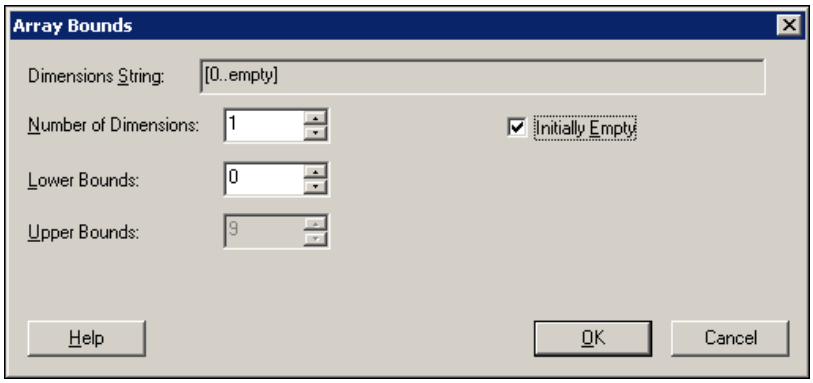


Figure 9-4. Array Bounds Dialog Box

e. Click **OK** to close the Array Bounds dialog box.

11. Rename the new Array property to `Measurement`. TestStand now can reference this array as `Step.Result.Measurement`.

End of Part A

Part B: Configuring the Properties Dialog Box

1. Use Windows Explorer to copy the `C:\Exercises\TestStand I\Waveform` directory to the `<TestStand>\Components\User\StepTypes` directory for use later in this exercise.
2. Right-click the `WaveformLimitTest` in the left pane and select **Properties** from the context menu to launch the Step Type Properties dialog box.
3. Enter the following information in the Step Type Properties dialog box to replace the current text in the fields:

   General Tab
   Default Step Name Expression: `"Waveform Limit Test"`

Step Description Expression: `"Tests the waveform's measured values against a high and low value"`

Menu Tab
Item Name Expression: `"Waveform Limit Test"`

Group: `Tests`

Figure 9-5 shows the resulting General Tab of the Step Type Properties dialog box.

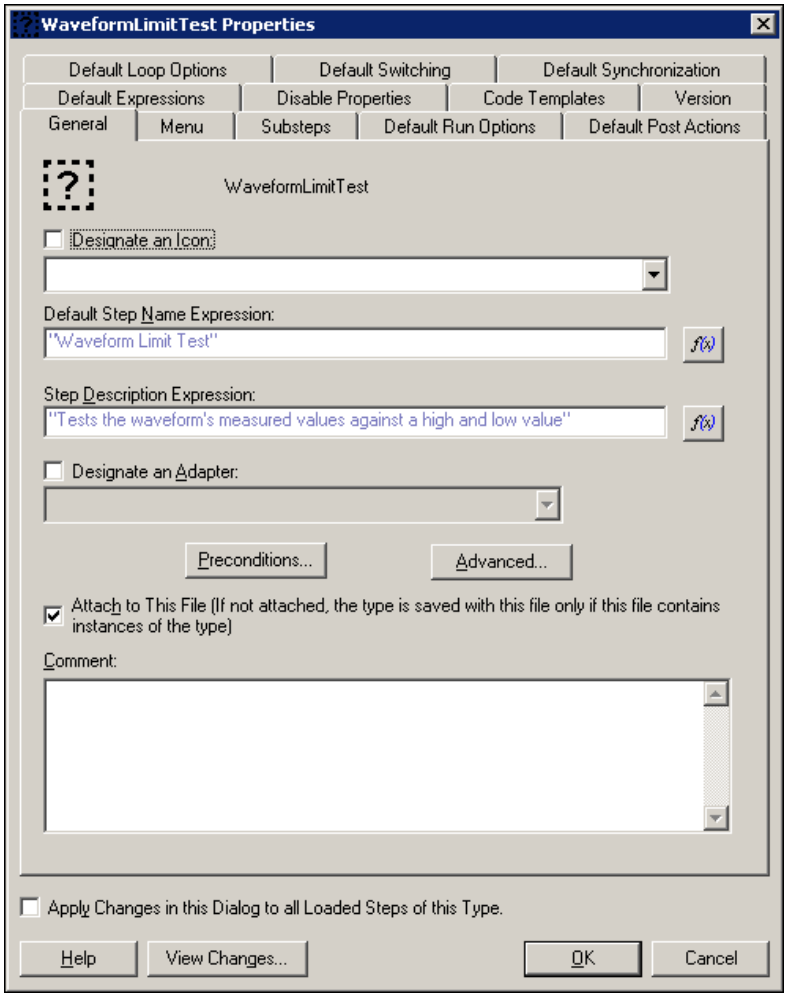


Figure 9-5. Step Type Properties Dialog Box

Now you need to implement two additional features in this step. Provide an Edit substep so the user can determine high limit and low limit values and create a Post-Step substep that can determine whether the step passed or failed based on the waveform data stored in the Measurement property.

4. In the Step Type Properties dialog box, select the **Substeps** tab.
5. Click **Delete** to delete the current Edit Substep step type.

   Create a custom Edit Substep step type using the LabVIEW Adapter.

6. Select the **LabVIEW Adapter** from the **Adapter** ring control. Click **Add** and select **Edit** from the context menu.
7. Change the **Menu Item Name Expression** to `"Edit Waveform Limits."`

   Additional Information
   By specifying this expression, when a user inserts a Waveform Limit Test into a test sequence and right-clicks the step, the context menu shows an item called Edit Waveform Limits.

Identify the VI that will run when the user selects the Edit Waveform Limits context menu item.

8. Click **Specify Module** to specify which VI is called when the user selects the Edit Waveform Limits item in the context menu.
9. Click **Browse** next to the **Path and Execution** box, and select `<TestStand>\Components\User\StepTypes\Waveform\editstep.vi`.

Note The VI for this substep is written for you. You copied the VI into the TestStand StepTypes directory in step 1.

10. If you see a VI Has Changed dialog box, this indicates that your version of LabVIEW is higher than the version used to save the VI. Choose **Yes** to open LabVIEW and then select **File»Save** and **File»Exit**. If you do not see this dialog, proceed to the next step.
11. Enable the **Show VI Front Panel When Called** option.

    Figure 9-6 shows the completed Edit LabVIEW VI Call dialog box for the Edit Waveform Limits.

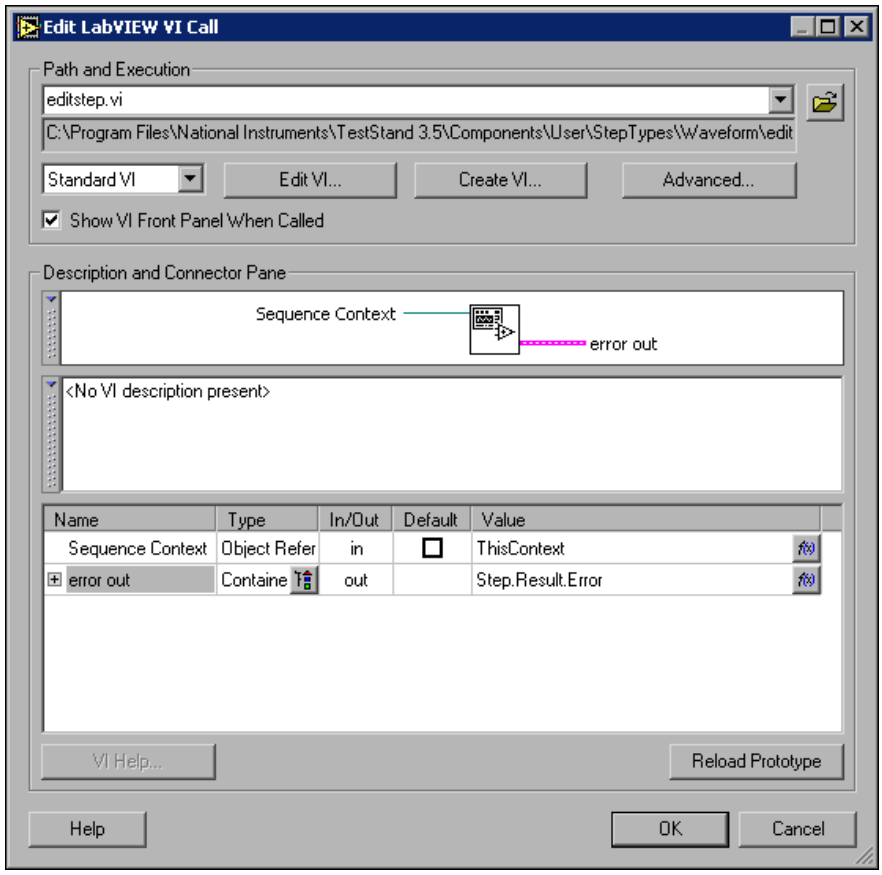


Figure 9-6. Edit LabVIEW VI Call Dialog Box

12. Click **OK** to close the Edit LabVIEW VI Call dialog box.
13. Click **Add** and select **Post-Step** from the context menu to add a Substep which performs the limit testing of the waveform data points to the declared limit values.
14. Click **Browse** next to the **Path and Execution** box, and select `<TestStand>\Components\User\StepTypes\Waveform\posttest.vi`.
15. If you see a VI Has Changed dialog box, this indicates that your version of LabVIEW is higher than the version used to save the VI. Choose **Yes** to open LabVIEW and then select **File»Save** and **File»Exit**. If you do not see this dialog, proceed to the next step.

    Figure 9-7 shows the Edit LabVIEW VI Call dialog box for the `Post Substep`.

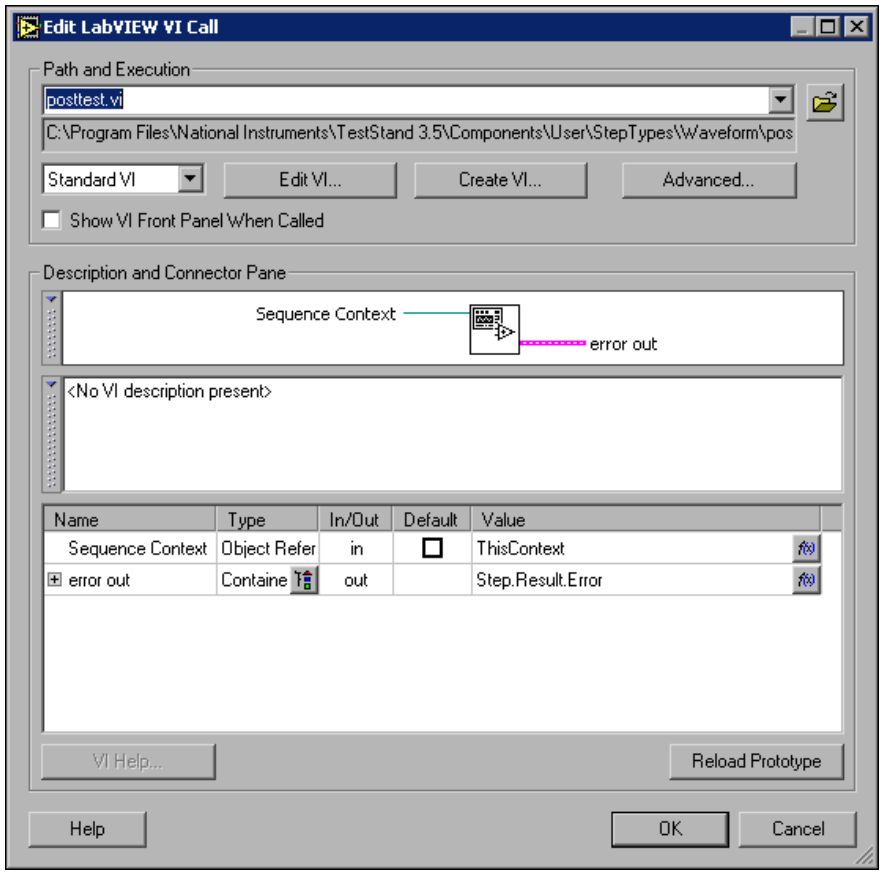


Figure 9-7. Edit LabVIEW VI Call Dialog Box for the Post Substep

16. Click **OK** to close the Edit LabVIEW VI Call dialog box.

Note If you are creating your own custom step types for later use, it is recommended that you create a new directory in the `<TestStand>\Components\User\StepTypes` folder to store the code modules that each custom step type uses. For example, in the case of the Waveform Limit Test, you copied the `C:\Exercises\TestStand I\Waveform` directory that contained the files, `editstep.vi` and `posttest.vi`, to the `<TestStand>\Components\User\StepTypes\Waveform` directory. Likewise, your paths for each substep pointed to these files. This maintains the default structure of the other NI installed types and makes distributing or sharing your test sequences easier.

Figure 9-8 shows the resulting Step Type Properties dialog box.

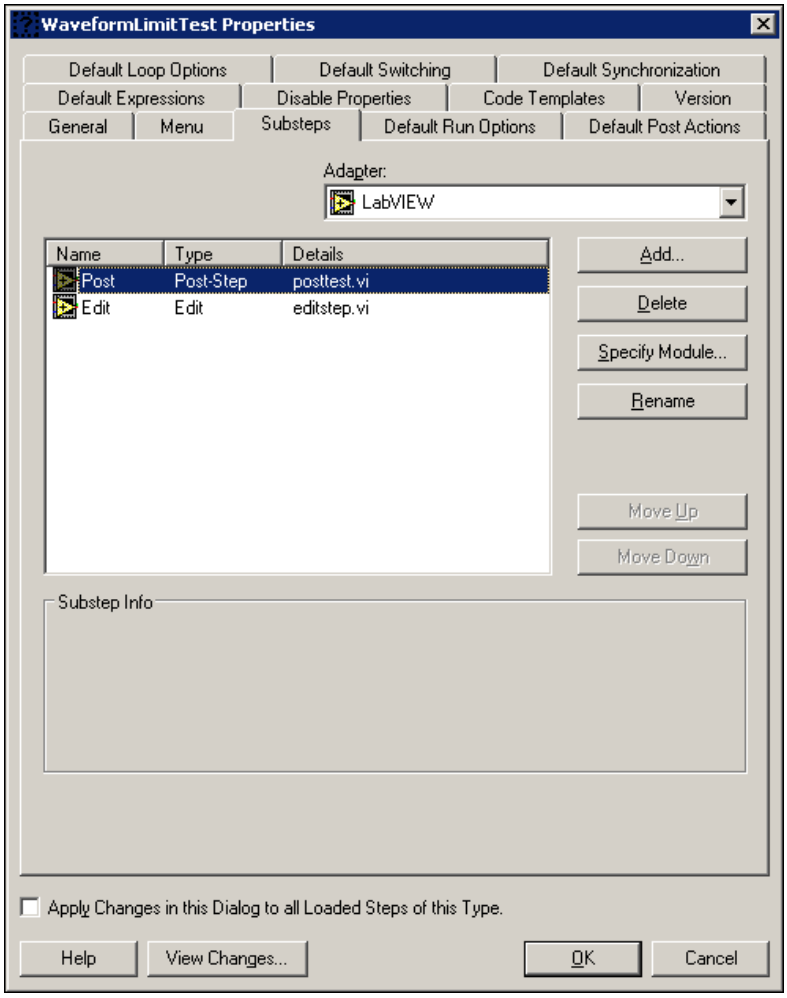


Figure 9-8. Step Type Properties Dialog Box

17. Because the limit tests are performed in a Post-Step substep, the default Status Expression is no longer needed to determine the Pass/Fail status of the step. Therefore, remove the expression from the Status Expression control. Select the **Default Expressions** tab in the Step Type Properties dialog box and delete the existing **Status Expression**. All three expression fields should be empty, as shown in Figure 9-9.

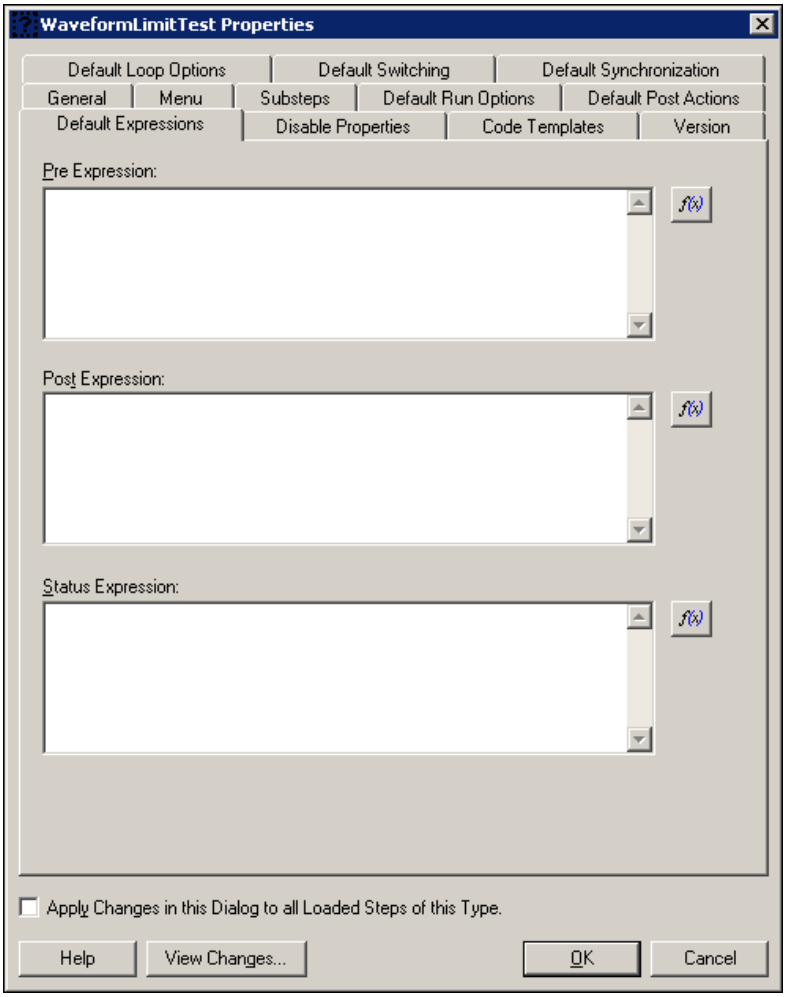


Figure 9-9. Step Type Properties Dialog Box Default Expressions Tab

18. Click **OK** to close the Step Type Properties dialog box.

You have modified a copy of the NumericLimitTest step type so that the custom step type performs a Waveform Limit Test. In the next exercise, you will use the Waveform Limit Test step in a test sequence.

End of Part B

End of Exercise 9-1

Exercise 9-2 Using a Custom Step Type

Objective: To use the Waveform Limit Test step created in Exercise 9-1 in a test sequence.

Part A: Use the Waveform Limit Test step in a Test Sequence

In this exercise, you will use the Waveform Limit Test step type that you created in Exercise 9-1.

1. Close all windows that are currently open inside the sequence editor application, including the Type Palette window.
2. Select **File»New Sequence File** to create a new sequence file. Save the sequence file as `Ex 9-2 Using the Waveform Limit Test.seq` in the `C:\Exercises\TestStand I` directory.
3. Select the **LabVIEW Adapter** from the **Adapter** ring control.
4. In the Sequence File window, right-click and select **Insert Step»Tests** from the context menu. The Waveform Limit Test appears in the context menu, as shown in Figure 9-1.

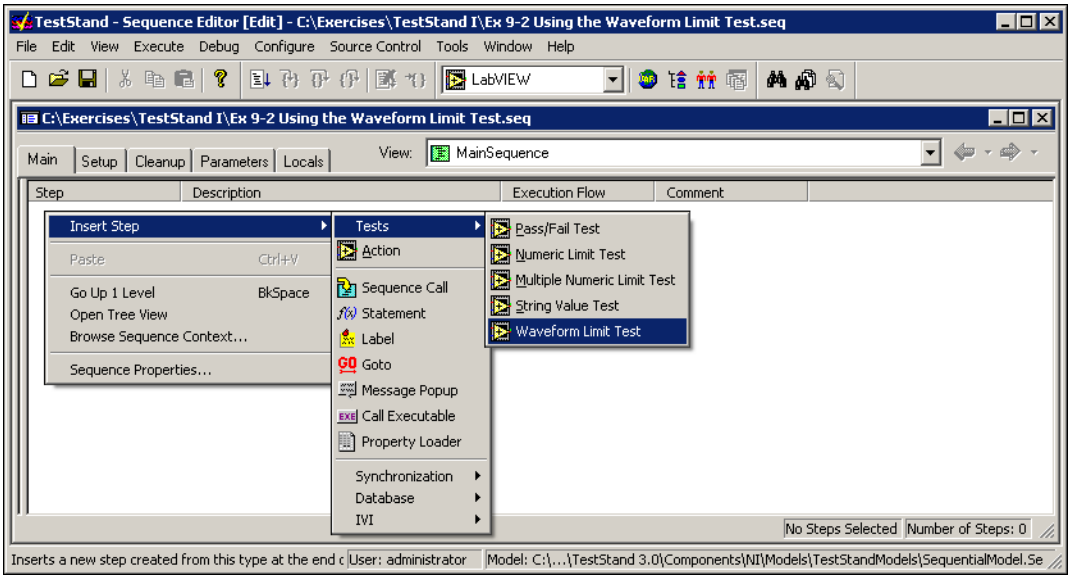


Figure 9-1. Context Menu Containing Waveform Limit Test

5. Insert a Waveform Limit Test step into the Main step group of the `MainSequence`. Rename the step `Sine Wave Test`.
6. Right-click the `Sine Wave Test` step and select **Specify Module** from the context menu to open the Edit LabVIEW VI Call dialog box.
7. Click **Browse** next to the **Path and Execution** box, and select `C:\Exercises\TestStand I\VIs\waveform.vi`. Select the **Use a relative path for the file you selected** option if the File not Found dialog box appears.

8. Enable the **Show VI Front Panel When Called** option.

   Figure 9-2 shows the resulting Edit LabVIEW VI Call dialog box.

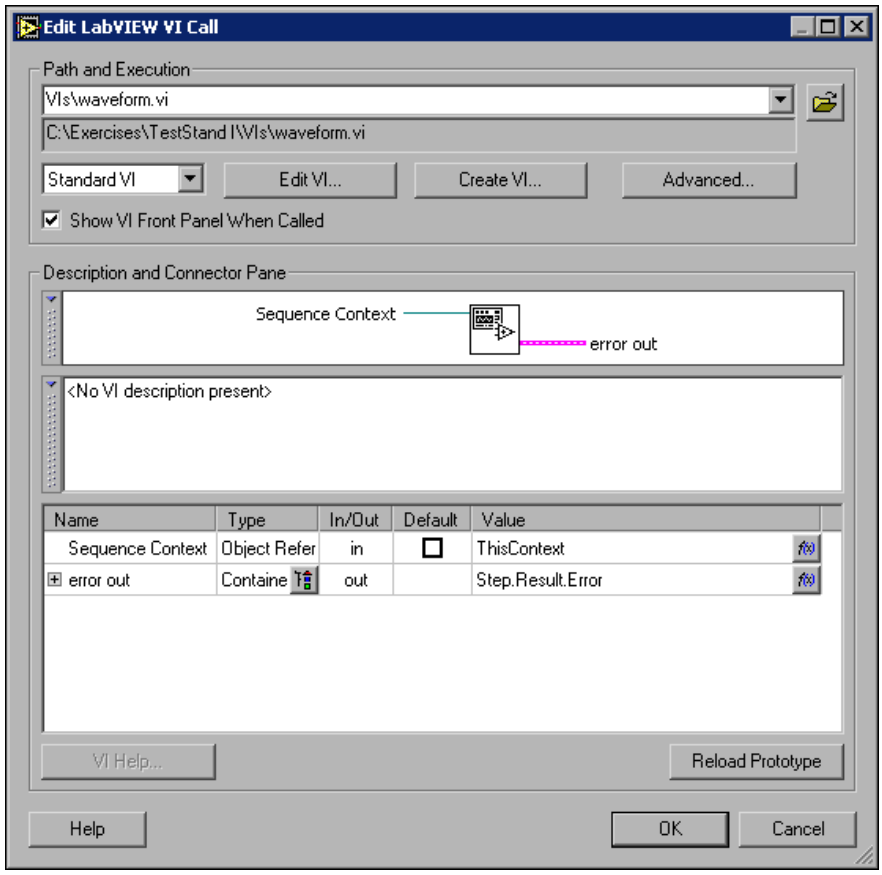


Figure 9-2. Edit LabVIEW VI Call Dialog Box

9. Click **Edit VI** to view the front panel and block diagram of the `waveform.vi`.

   Notice in the block diagram, shown in Figure 9-3, that after generating the sine wave array, the VI passes the waveform data into the property Step.Result.Measurement using the Numeric Array instance of the Set Property Value VI.

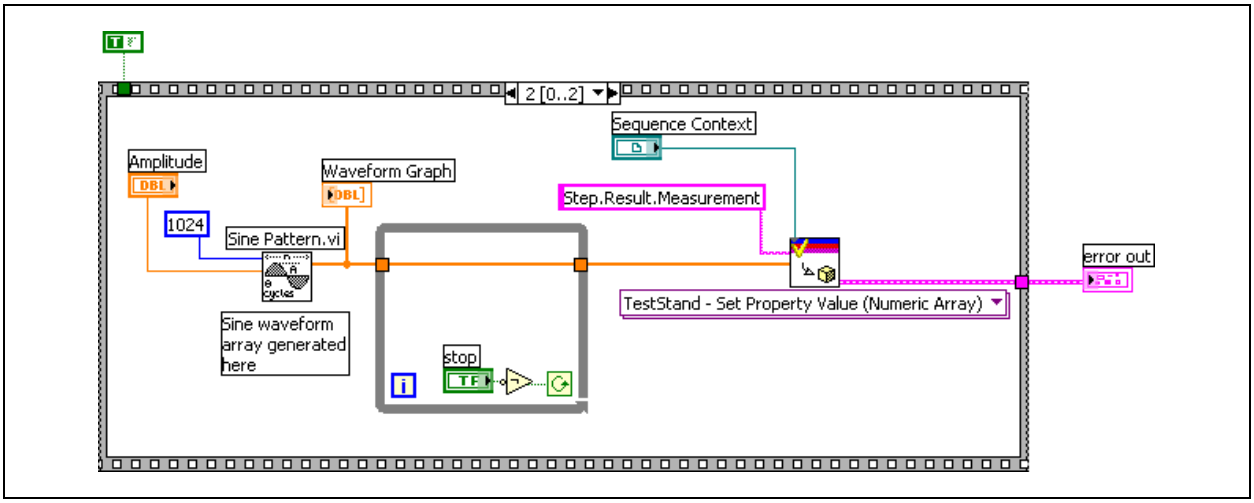


Figure 9-3. Waveform.vi Block Diagram

10. After reviewing the block diagram, close the VI.
11. Click **OK** to close the Edit LabVIEW VI Call dialog box.
12. Right-click the `Sine Wave Test` step and select **Edit Waveform Limits** from the context menu. This displays the VI front panel you specified in the Edit substep for the Waveform Limit Test step type in Exercise 9-1.

    Notice that the editstep.vi dialog box, shown in Figure 9-4, displays the current high and low limits. You can change these limits and click **OK** to store the changes.

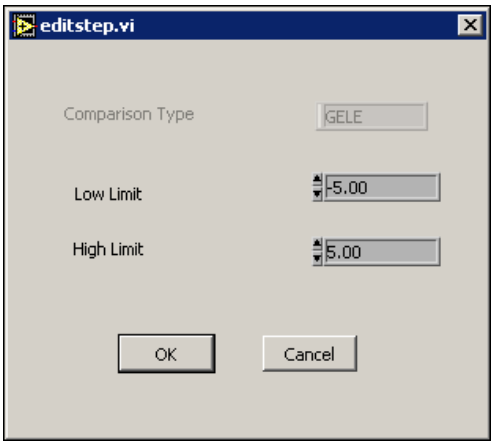


Figure 9-4. Editstep.vi Dialog Box

Additional Information

Notice that the Comparison Type field is disabled in this exercise. This field indicates what kind of comparison TestStand will perform on the waveform measurements. The only comparison type that is built into this example is GELE (low limit <= measurement <= high limit). You

can add other comparison types by modifying the `editstep.vi` and the `posttest.vi`.

13. Save the changes to the sequence file.
14. If a type conflict Warning dialog box appears, as shown in Figure 9-5, enable the **Save these settings and do not prompt again** option and click **OK**.

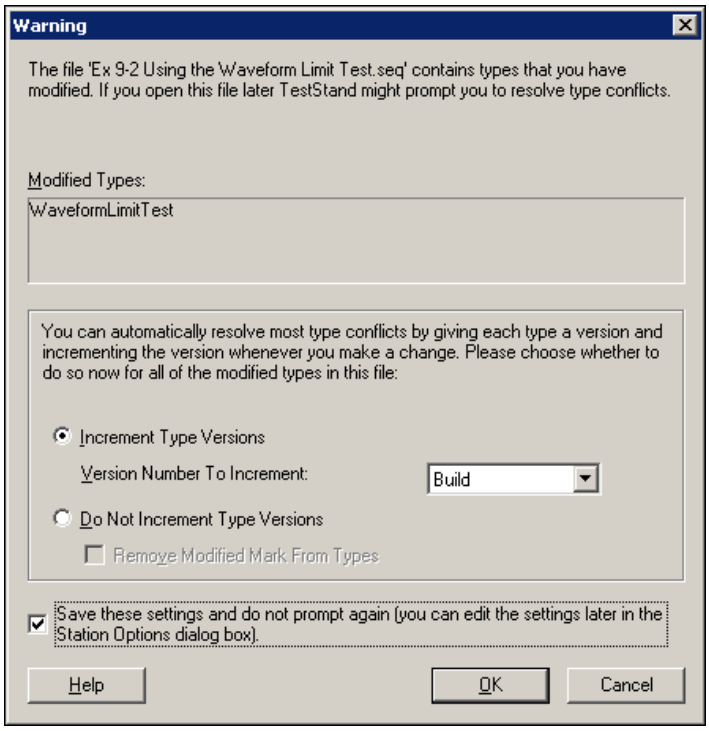


Figure 9-5. Type Conflict Warning

15. Select **Execute»Single Pass** to execute the sequence file

    When the test runs, the Sine Wave Test step displays a VI front panel that prompts you to enter the amplitude for the sine wave.
16. Click **OK** to generate the sine wave with the given amplitude. The graph in Figure 9-6 shows the generated sine wave.

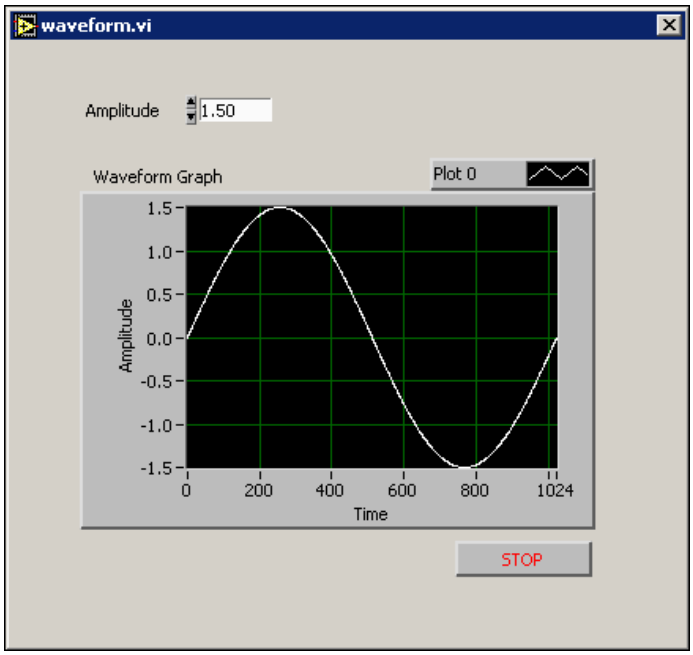


Figure 9-6. Generated Waveform Graph

17. Click **Stop** to complete the test and send the waveform data back to the Step.Result.Measurement property in TestStand for limit comparison.

    Notice that if the amplitude is higher than the high limit, the report indicates that the step failed. If the amplitude is set within the limits, the test passes.

End of Part A

Part B: Updating the HTML Report to Show Waveform Data

Now that you have created a test to handle and compare your waveform data, modify the step to display the waveform in the test report. In this part, you will make a few modifications to the Waveform Limit Test step so that it displays the waveform in the report.

1. Select **Configure»Report Options** to open the Report Options dialog box.
2. Confirm that the **Include Measurements** option is enabled and the **Include Arrays** option is set to **Insert Graph**. Set the **Filter** option to **Include All**.

   Figure 9-7 shows the Report Options dialog box.

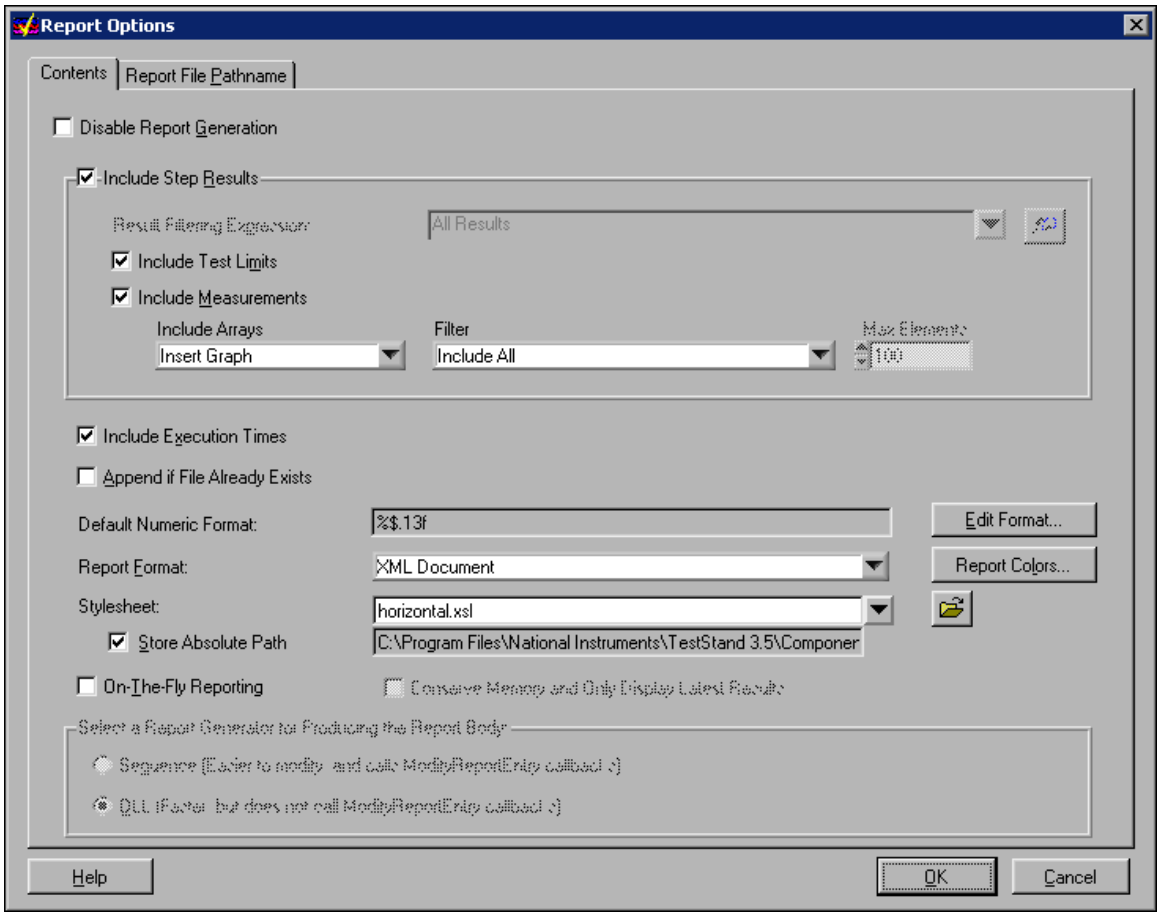


Figure 9-7. Report Options Dialog Box

3. Click **OK** to close the Report Options dialog box.
4. Click **Type Palette** on the sequence editor toolbar to open the Type Palette window.
5. Click the **+** in front of the `WaveformLimitTest` in the left pane of the Type Palette window to expand the `WaveformLimitTest` properties.
6. One of the properties listed under the `WaveformLimitTest` is the `Result` property. Highlight the `Result` property in the left pane to show the subproperties within it. The right pane shows all the subproperties contained inside the `Result` property.
7. Double-click the `Measurement` property in the right pane and click **Advanced** in the Measurement Properties dialog box to open the Edit Flags dialog box.
8. Enable the **PropFlags_IsMeasurementValue** and **PropFlags_IncludeInReport** flags.
9. Enable the **Reset Flags in All Loaded Instances of the Type** option. Figure 9-8 shows the resulting Edit Flags dialog box.

Note Enabling the Reset Flags in All Loaded Instances of the Type option for the Waveform Limit Test step updates the flags that you are using in the test sequence. These flags inform TestStand that the Numeric Array property, `Measurement`, is a measurement that is obtained from the step and that it should be included in the test report.

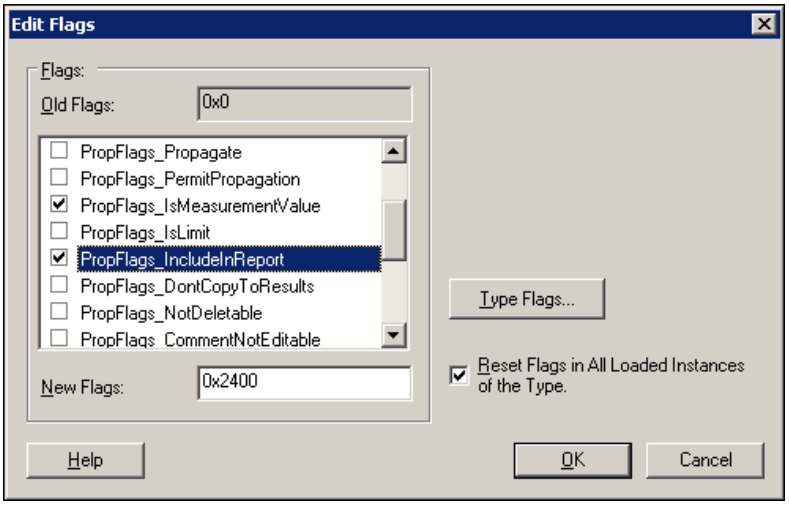


Figure 9-8. Edit Flags Dialog Box

10. Click **OK** to close the Edit Flags dialog box.
11. Click **OK** to close the Measurement Properties dialog box. You can also save the changes and close the Type Palette window.
12. Open the `Ex 9-2 Using the Waveform Limit Test.seq` file located in the `C:\Exercises\Teststand I` directory.
13. Select **Execute»Single Pass to** run this sequence.
14. Run the test by adjusting the amplitude. When the test is done, notice that the HTML report now contains the waveform measurement in the graph, as shown in Figure 9-9.

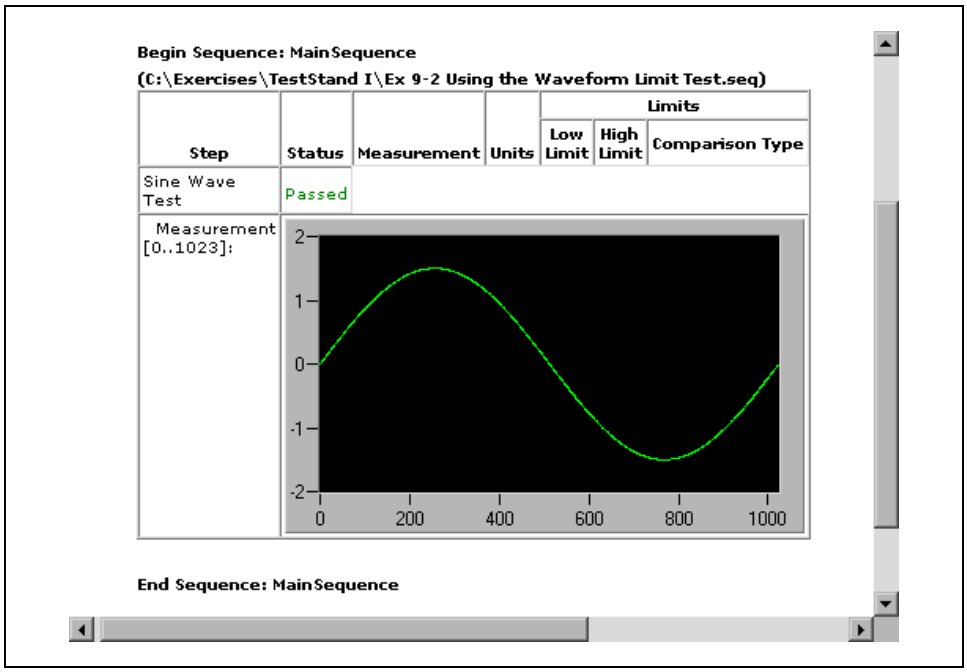


Figure 9-9. Report with Included Graph

End of Part B

End of Exercise 9-2

Self Review

1. What are the four windows used for viewing TestStand types, and how do they differ in scope?

2. When configuring the step properties of your custom step type, what does the Default Step Name Expression configuration on the General tab correspond to?

3. List the following in terms of their execution order, starting with the first executed component: Code Module, Precondition, Post-Step Substep, Pre-Step Substep, Status Expression, Pre-Expression, and Post-Expression.

Notes

Notes

10

Database Interaction

Lesson 10: Database Interaction

In this lesson, you will:

- **Enable database logging in TestStand**
- **Use the database viewer to monitor the contents of a database**

Introduction

TestStand can log UUT results and step results to a database. This lesson describes the database interaction features in TestStand. You will learn how to enable TestStand to log results to an Access database by default and use the Database Viewer to view data that TestStand has logged to the database.

Database Concepts

The table shows an example of the following database concepts:

- Database
- Tables
- Records (rows)
- Fields (columns)

Sample Test Sequence Results			
UUT_NUM	**STEP_NAME**	**RESULT**	**MEAS**
20860B456	**TEST1**	**PASS**	**0.5**
20860B456	**TEST2**	**PASS**	**(NULL)**
20860B123	**TEST1**	**FAIL**	**0.1**
20860B789	**TEST1**	**PASS**	**0.3**
20860B789	**TEST2**	**PASS**	**(NULL)**

Database Concepts

The following database concepts are important for understanding how TestStand interacts with databases.

- **Database**—An organized collection of data. You can store data in and retrieve data from a database. The data is usually stored in a table.
- **Table**—Most modern Database Management Systems (DBMS), also called database servers, store data in table form. There can be multiple tables in a database. Each table in a database must have a unique name.
- **Record**—Similar to a row in a matrix. The term record is often interchanged with the term row since they are similar in meaning.
- **Field**—Similar to a column in a matrix. Each field in a database table must have a unique name.

The example above shows results from a test sequence. The fields (columns) are `UUT_NUM`, `STEP_NAME`, `RESULT`, and `MEAS`. Each record or row represents a different Unit Under Test (UUT). Some entries in the `MEAS` field hold a `NULL` value. This empty field value is also referred to as an SQL Null value.

Databases and Database Drivers

- **DBMS supported by default:**
  - Microsoft Access
  - Microsoft SQL Server
  - Oracle
  - MySQL
  - Sybase Adaptive Any where
- **Support for other DBMS can be added by installing drivers to communicate with your DBMS**
  - ODBC Driver»Data Source Name (DSN)
  - Specific OLE-DB Provider

Databases and Database Drivers

Before you can access data from within TestStand, you must provide a data link that contains information about the database interface, such as the provider, the server on which the data resides, the database or file that contains the data, the user ID, and permissions to request when connecting to the data source.

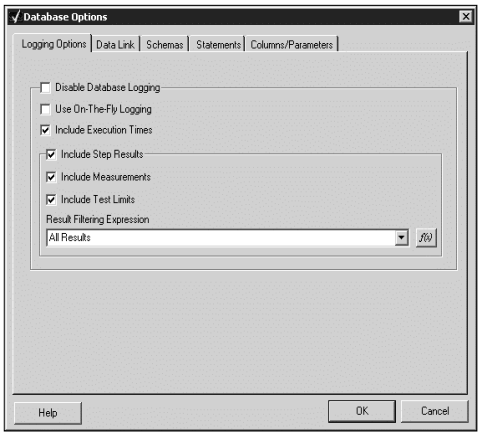
By default, TestStand supports Oracle, Microsoft Access, and Microsoft SQL Server. If you decide to use another DBMS, refer to the *Adding Support for Other Database Management Systems* section in Chapter 6, *Database Logging and Report Generation*, of the *TestStand Reference Manual* for more information. By default, TestStand supports Oracle, Microsoft Access, Microsoft SQL Server and others, but TestStand does not install the drivers for these systems, or distribute the systems to target machines. Refer to the vendor documentation for information about registering your specific database drivers with the ODBC Administrator.You must decide whether to use an ODBC driver or a specific OLE-DB Provider for your DBMS. Refer to the `<TestStand>\Doc\readme.txt` for suggested versions of client DBMS software.

When you use the Microsoft OLE DB provider for ODBC Drivers to connect to an ODBC data source, you must first configure a data source name (DSN) within the ODBC Administrator available through the Windows Control Panel. The DSN specifies which ODBC driver to use, the database file (`.mdb`), and the optional user ID and password.

If you are using a specific OLE-DB Provider for your DBMS, you can link directly to the corresponding database from within TestStand.

Default Logging to Database

Configure»Database Options»Logging Options

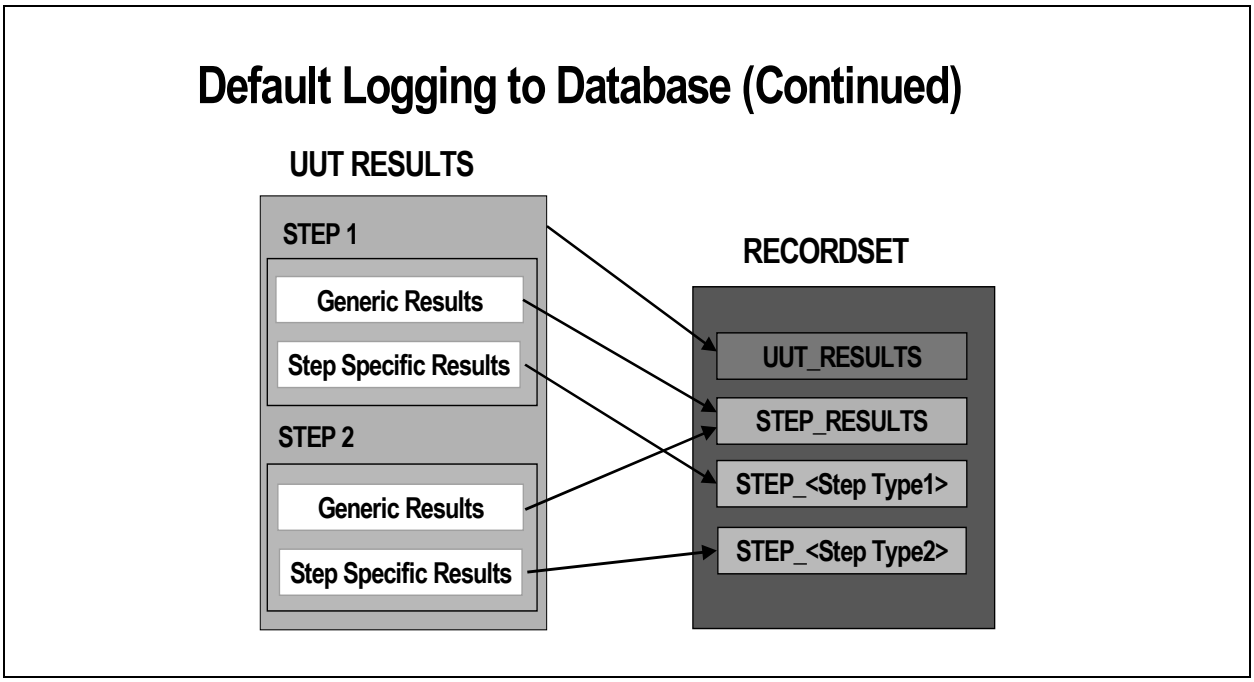


Default Logging to Databases

To log the results of your UUT to a database, select **Configure»Database Options**. TestStand executes the Configure Database Options Configuration entry point (purple icon) in the process model.

The Logging Options tab in the Database Options dialog box allows you to choose whether the test results are logged to a database and specify what default step data is logged to the database. This includes adding filtering expressions to add inclusion criteria for database logging.

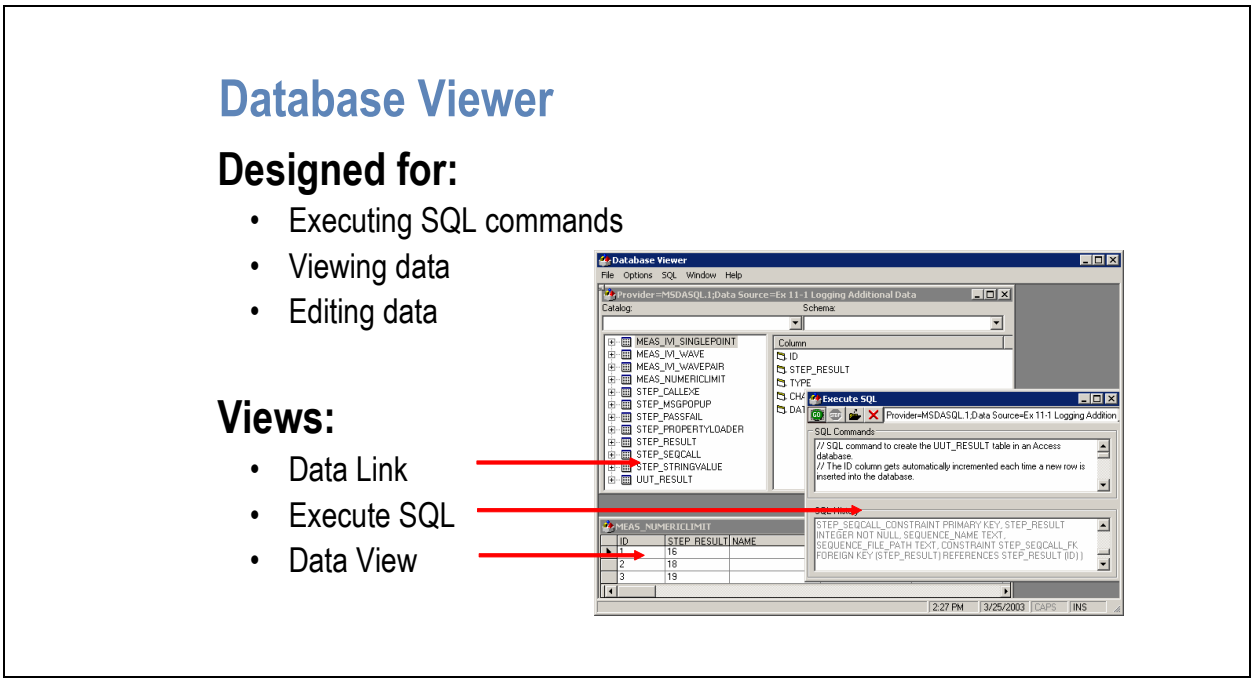
The remaining tabs allow you to link to a specific database and customize how your test system results are logged to the database. Refer to Lesson 11, *Custom Database Logging*, for more information.



Default Logging to Database (Continued)

To log your results, TestStand uses a schema that defines what results are logged to your database and into which table and field each result is logged. The default schema is configured so data pertaining to UUT results, such as the UUT status and UUT serial number, log to the `UUT_RESULT` table in your database. The generic results of every step, such as Step Status and Step Name log to the `STEP_RESULTS` table. The step results that are unique for each step type log to separate tables, one table for each step type. For example, a Numeric Limit Test step will have high and low limit results, which log to the `STEP_NUMERICLIMIT` table. Similarly, data that is specific to the String Value Test step, such as the Comparison String property logs to a table called `STEP_STRINGVALUE`.

Keep in mind that these tables are defined by the default schema that TestStand uses. You might define your own schema to control how TestStand logs your data. Refer to Lesson 11, *Custom Database Logging*, for more information about controlling how TestStand logs data.



Database Viewer

Use the Database Viewer to create, edit, or view database tables and data. Click **View** on the **Data Link** tab of the Database Options dialog box and launch the Database Viewer or select **Tools»Database Viewer**. The Database Viewer displays the following three types of windows:

- **Data Link window**—Contains a tree view of the tables that are defined for the data link. The list view displays the contents of the currently selected node in the tree view. Right-click in the window, to display a context menu. The items in the context menu vary depending on whether you right-click a node in the tree view, and entry in the list view, or the background area. The context menu can contain the following items:
  - **View Data**—Opens a new Data View window with the contents of the table.
  - **Add Table**—Creates a new table in the DBMS.
  - **Drop Table**—Deletes a table from the DBMS.
  - **Add Column**—Adds a new column to a table.
  - **Drop Column**—Deletes a column from a table.
- **Execute SQL window**—Contains a SQL Commands control and a SQL History control. You can enter SQL commands in the SQL Commands control and to execute its contents, click **Go**. Click **Open File**, to load SQL script files. Use the Clear button to delete the contents from the SQL Commands control.
- **Data View window**—Contains a grid display of the data returned from a SQL command. The Database Viewer automatically opens a new Data

View window when you select **View Data** from the context menu or when you issue a SQL command that returns a recordset. You can also edit data in a table using this window.

Database Demo

1. Create a sequence that contains a Numeric Limit Test step, a String Value Test step, and a Pass/Fail Test step.
2. Configure TestStand to log your results to the default database.
3. Run the Database Viewer and verify that the data has been logged.

Database Demo

Exercise 10-1: Logging UUT Results to a Database

Objective: To log UUT results to a Microsoft Access database.

Estimated Time: 30 minutes

Refer to page 10-11 for instructions for this exercise.

Lesson 10: Summary

- Log results to a database by selecting Configure»Database Options and enabling database logging
- View data in the database using the Database Viewer application

Summary

In this lesson, your learned how to log your results to the default database in TestStand. To enable database logging, you must select **Configure» Database Options** and remove the checkmark from the **Disable Database Logging** option. Once you test your UUT, you can run the Database Viewer application to view the data in the database. Launch the Database Viewer by selecting **Tools»Database Viewer**.

Exercise 10-1 Logging UUT Results to a Database

Objective: To log UUT results to a Microsoft Access database.

This exercise outlines how to setup a TestStand data link to a Microsoft Access database file (`*.mdb`) to log results using the default Sequential process model. This exercise uses the OLE DB provider for ODBC. The Microsoft Access ODBC driver connects to the data source. When accessing a database using an ODBC driver, you must create a Data Source Name (DSN). Create a DSN using the Windows ODBC Data Source Administrator.

Complete the following steps to set up a TestStand datalink to an Access database.

1. Verify that the Microsoft Access ODBC Driver is installed on your system.
2. Setup an ODBC Data Source Name (DSN).
3. Create a corresponding database file.

Part A: ODBC Data Source Administrator

In this section of the exercise, use the ODBC Data Source Administrator to create a DSN and the corresponding database.

1. Select **Start»Settings»Control Panel»Administrative Tools**.
2. Double-click the **Data Sources (ODBC)** icon to open the ODBC Data Source Administrator dialog box, shown in Figure 10-1.

Note If you are using older versions of Windows, select **Start»Settings»Control Panel** and double click on the **ODBC** icon to open the ODBC Data Source Administrator.

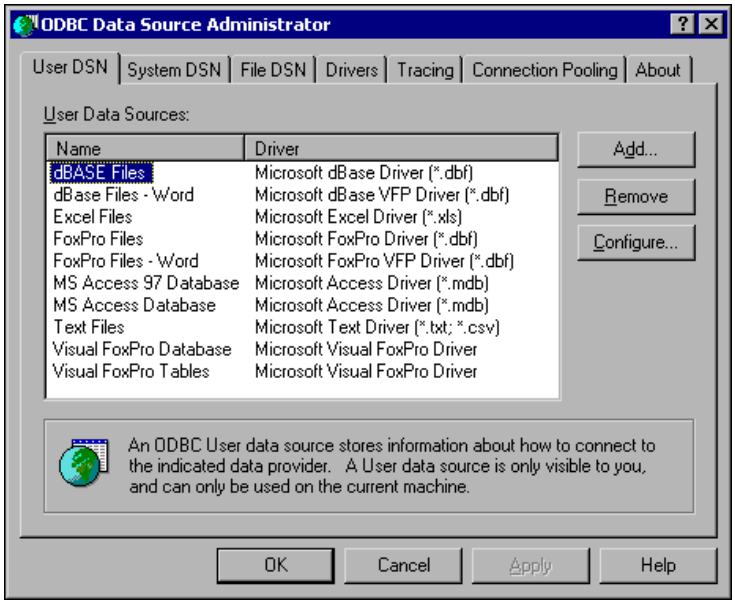


Figure 10-1. ODBC Data Source Administrator

3. Click the **Drivers** tab and verify that the **Microsoft Access Driver (*.mdb)** is installed on your computer.

   The **Drivers** tab and required driver are shown in Figure 10-2.

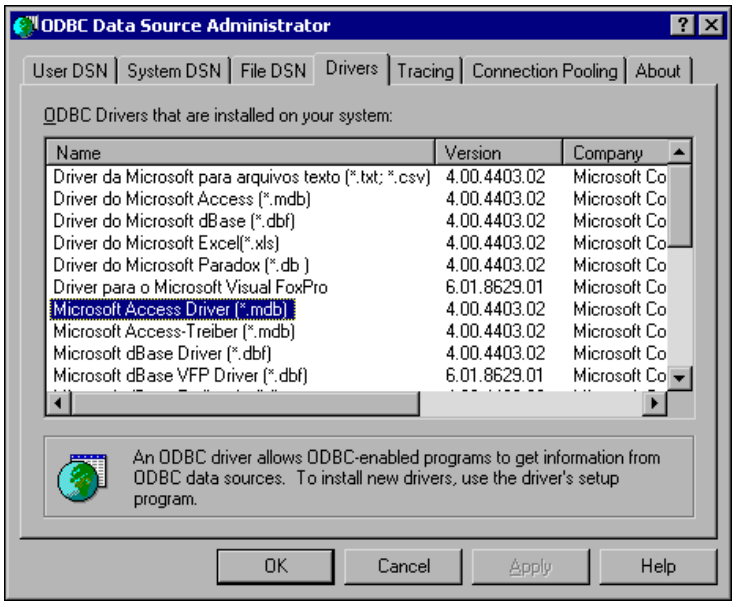


Figure 10-2. Data Source Administrator Driver Tab

4. Click the **User DSN** tab, shown in Figure 10-3, and select **MS Access Database**.

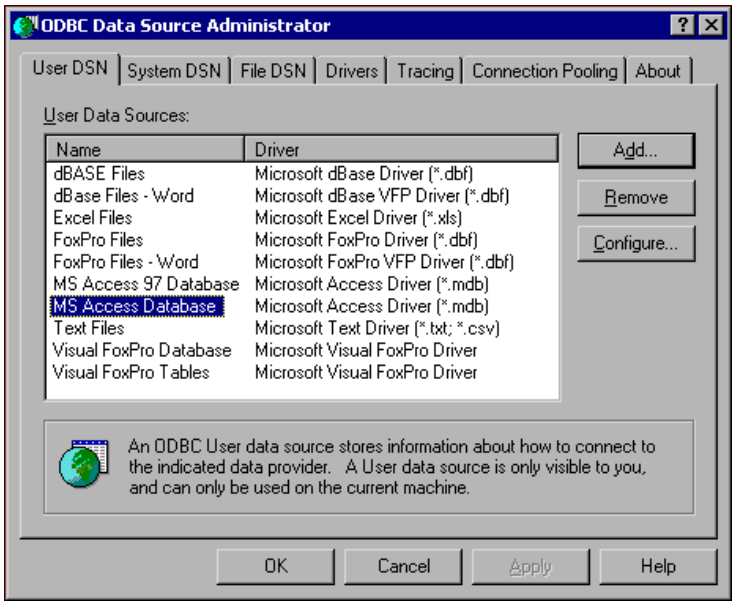


Figure 10-3. Data Source Administrator User DSN Tab

5. Click **Add** to launch the Create New Data Source dialog box.
6. Select **Microsoft Access Driver (*.mdb)** from the list. Figure 10-4 shows the resulting Create New Data Source dialog box.

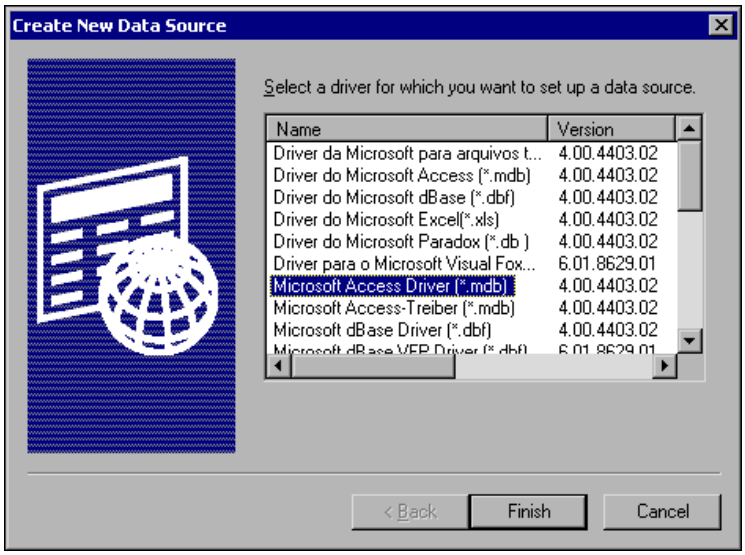


Figure 10-4. Create New Data Source Dialog Box

7. Click **Finish** to close the Create New Data Source dialog box.
8. Enter `TestStand Access` into the **Data Source Name** text field in the ODBC Data Source Administrator dialog box.

Figure 10-5 shows the resulting ODBC Microsoft Access Setup dialog box.

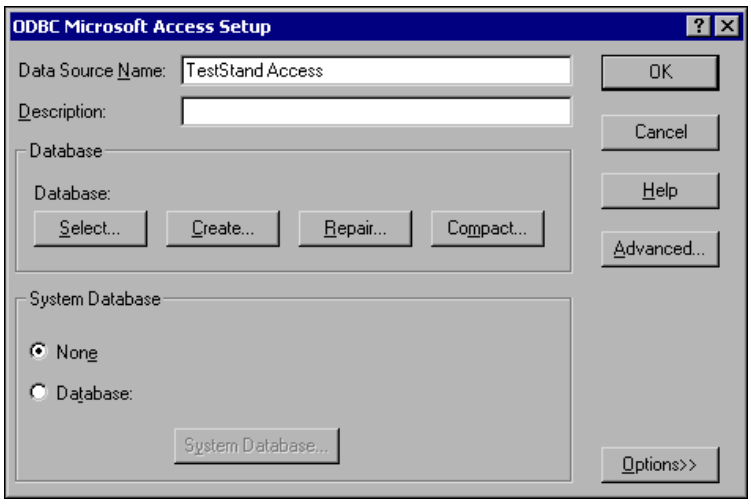


Figure 10-5. ODBC Microsoft Access Setup Dialog Box

9. Click **Create** in the **Database** section.
10. Navigate to the `C:\Exercises\TestStand I` directory and enter `Sequence Data.mdb` in the **Database Name** text field.

    Figure 10-6 shows the resulting New Database dialog box.

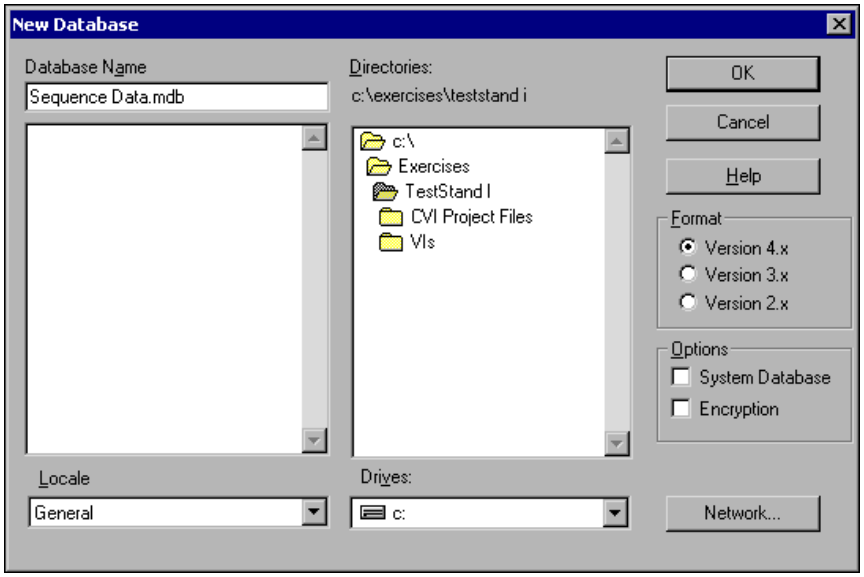


Figure 10-6. New Database Dialog Box

11. Click **OK** to close the New Database dialog box. A dialog box displays to inform you that the database was created successfully.

12. Click **OK** to close the ODBC Microsoft Access Setup and ODBC Data Source Administrator dialog boxes.

Note The steps you performed in this part of the exercise are only for creating a data source name and a database file. Once you have created these items, you do not have to repeat steps 1 through 12 to log results to a database in TestStand.

End of Part A

Part B: Database Options

After you create the ODBC data source, you must configure the database logging options in TestStand. To use the database logging features in TestStand, you must enable the database logging feature and configure certain parameters. In this part of the exercise, you enable the database logging features and configure the parameters.

1. Select **Configure»Database Options** to display the Database Options dialog box shown in Figure 10-7.

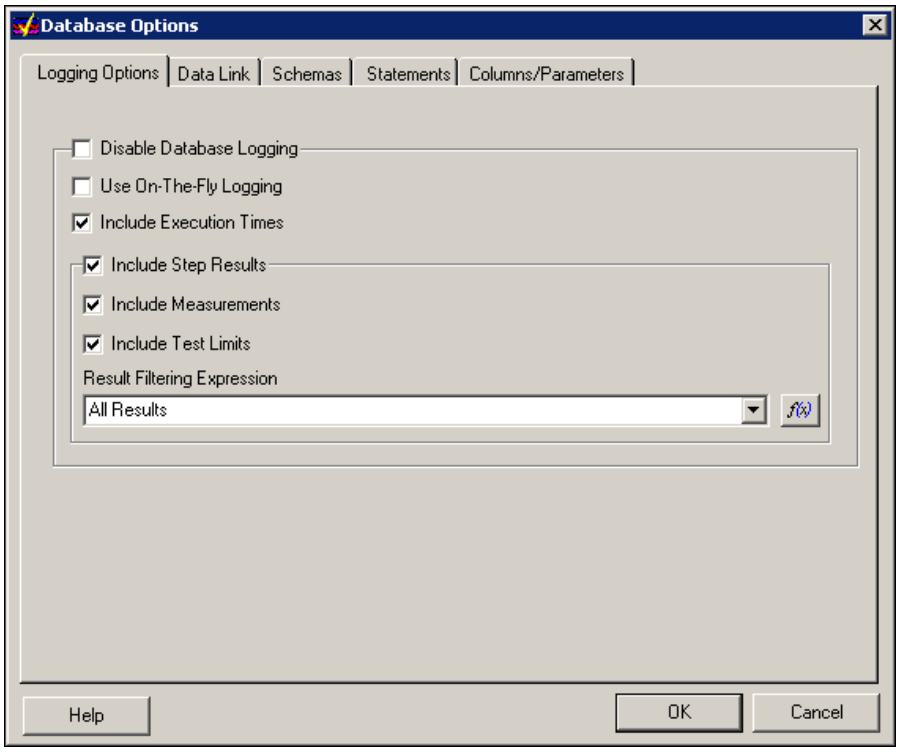


Figure 10-7. Database Options Dialog Box

2. Disable the **Disable Database Logging** option on the **Logging Options** tab.

Disabling this option enables the remaining options on the Logging Options tab. You configure these options in much the same way as you configure report options.

3. Click the **Data Link** tab. Use this tab to create a data link for TestStand to use.
4. Identify the data source name and database where you want to log the results.
   a. Verify that **Access** is selected in the **Database Management System** ring control.
   b. Select **Build** to create the **Connection String Expression**.
   c. Click the **Provider** tab of the Data Link Properties dialog box.
   d. Select **Microsoft OLE DB Provider for ODBC Drivers** as shown in Figure 10-8.

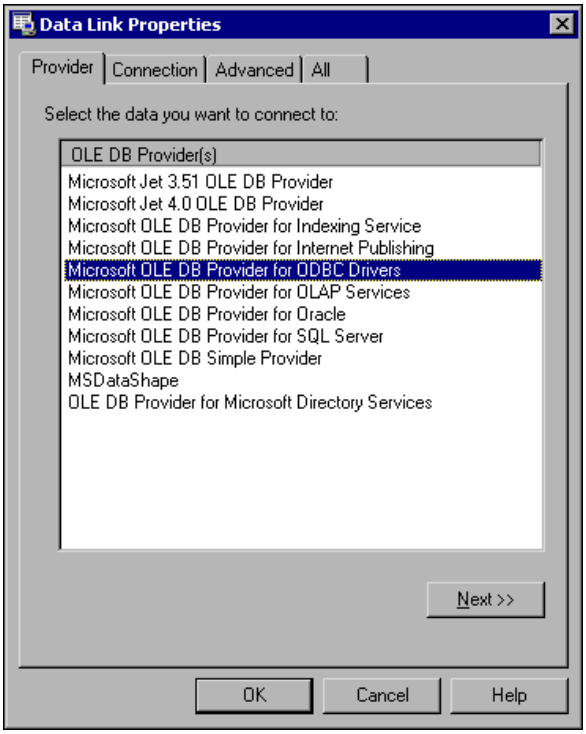


Figure 10-8. Data Link Properties Provider Tab

   e. Click the **Connection** tab.
   f. Select **TestStand Access** from the **Use data source name** ring control as shown in Figure 10-9.

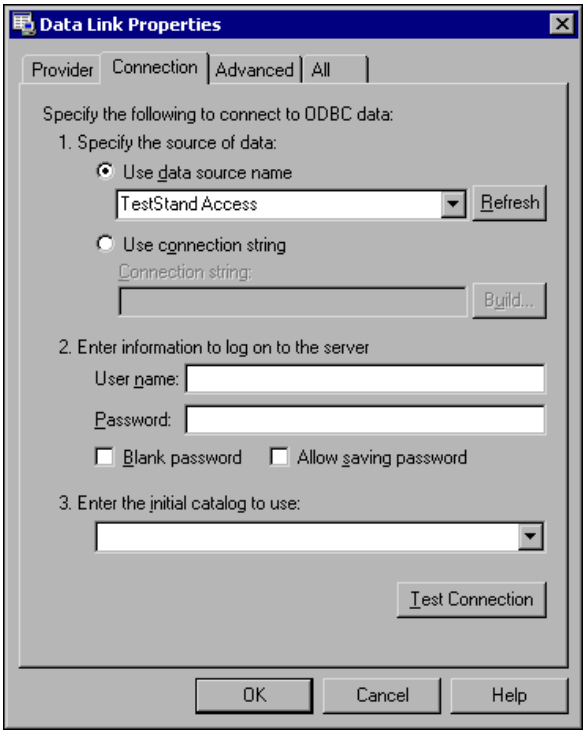


Figure 10-9. Data Link Properties Provider Tab

g. Click **Test Connection** to verify that TestStand can establish the connection.

h. Click **OK** to close the Data Link Properties dialog box.

5. The **Connection String Expression** should automatically update after you close the Data Link Properties dialog box. Figure 10-10 shows the resulting Data Link tab.

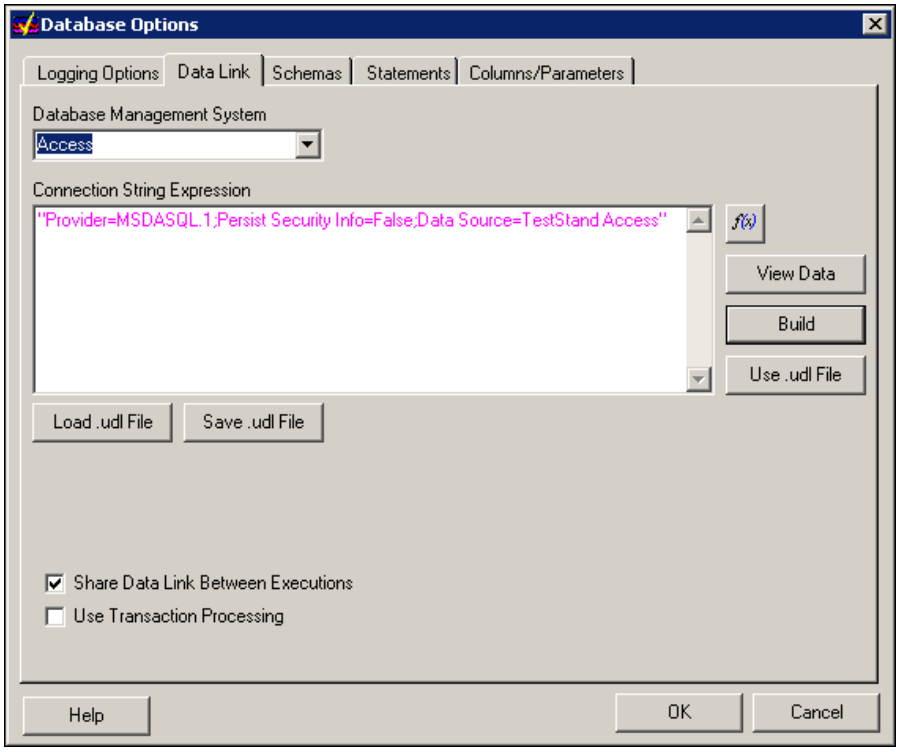


Figure 10-10. Database Options Data Link Tab

6. Click the **Schemas** tab and enable the **Generic Recordset (NI)** schema in the schema section.

   Additional Information

   Because you are using the default Generic Recordset (NI) schema, you do not need to configure the following two tabs of the Database Options dialog box.

   - **Statements**—Defines the data that TestStand logs.
   - **Columns/Parameters**—Specifies the columns or parameters that TestStand logs for each result for which the statement applies.

   The default schemas already contain this information. Figure 10-11 shows the Schemas tab.

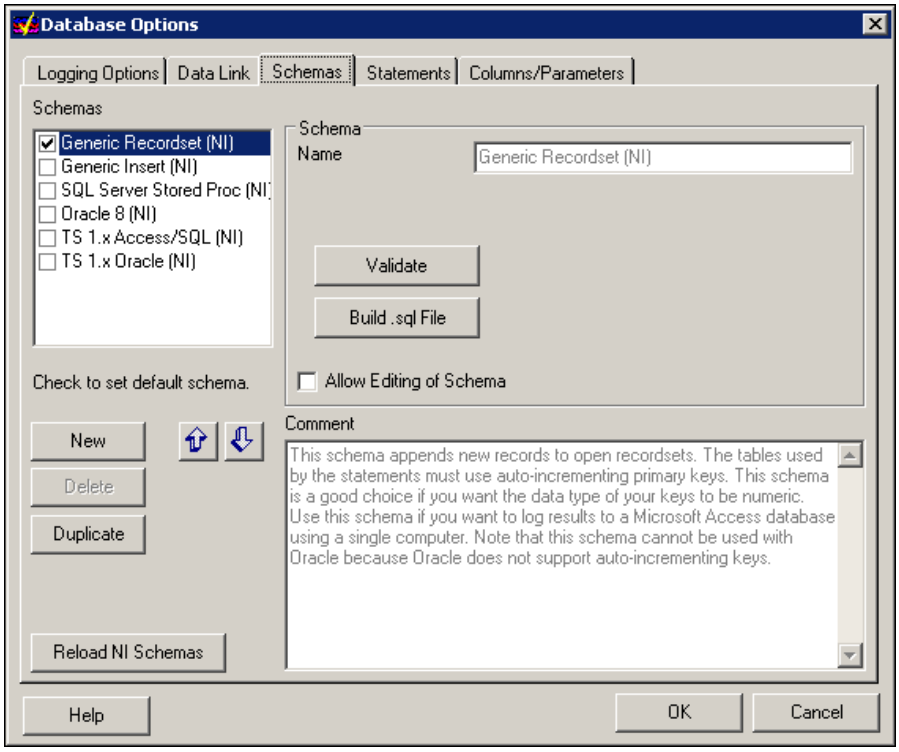


Figure 10-11. Database Options Dialog Box Schemas Tab

7. Click **OK** to close the Database Options dialog box.

End of Part B

Part C: Creating Database Tables

TestStand includes the Database Viewer application for viewing data, editing database table properties, and executing SQL commands in a database. The Database Viewer application, `DatabaseView.exe`, is located in the `<TestStand>\Components\NI\Tools\DatabaseView` directory.

In this part of the exercise, you will execute SQL commands to create the required database tables in the `Sequence Data.mdb` file you created. The `UUT_RESULT` table contains a record for each UUT that TestStand tests. The table includes fields for the UUT serial number, name of the test operator, date and time of the test, and the resulting pass/fail status of the UUT as well as others. The `STEP_RESULTS` table contains the results for the most common types of step result properties such as the step status and error information. The remaining tables, such as `STEP_MSGPOPUP`, store the specific types of result information for each step type.

1. Select **Configure»Database Options** to open the Database Viewer

Note You could also open the Database Viewer by selecting **Tools»Database Viewer**. In this exercise, use the method described in step 1.

2. Select **View Data** from the **Data Link** tab in the **Database Options** dialog box. Figure 10-12 shows the Database Viewer window that appears.

   Additional Information
   By selecting **Configure»Database Options**, you do not have to specify the data link information again because it is already stored in the Database Options dialog box.

   The Database Viewer window contains a tree view of the tables that exist in the database specified by the data link. This window is empty because you have not yet created tables within your database. The title of the Data Link window, `Provider=MSDASQL.1; Data Source=TestStand Access`, is a portion of your connection string.

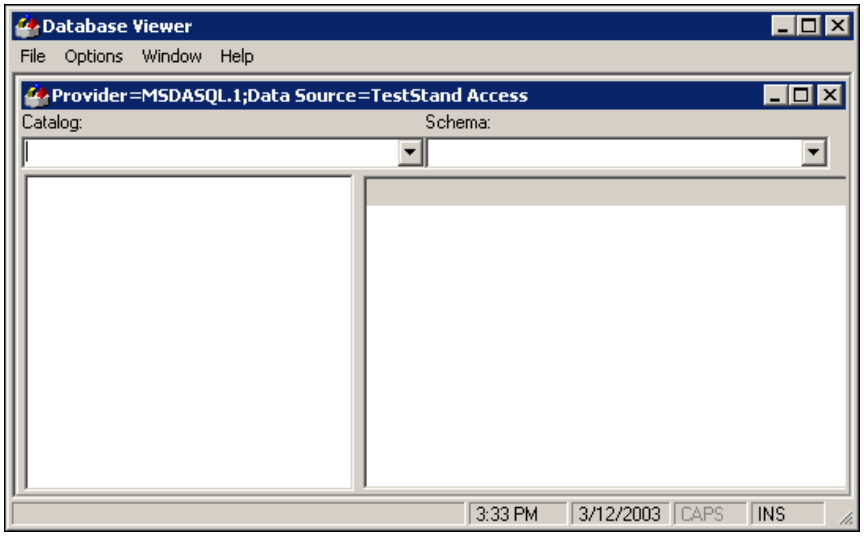


Figure 10-12. Database Viewer Application

3. Select **File»New Execute SQL Window** to begin creating tables in the database. The Execute SQL window should appear, as shown in Figure 10-13.

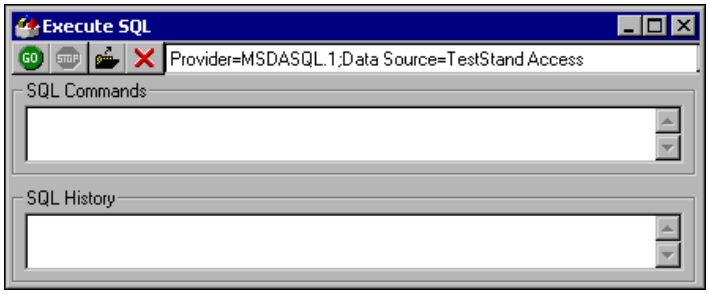


Figure 10-13. Execute SQL Window

4. Create a SQL script that creates the database table for you.

   TestStand installs SQL script files for creating and deleting the default database tables in Access, Oracle, SQL Server and other databases according to the default schemas. You can load and execute these ASCII text script files in the Execute SQL window.

   a. Click **Open File** on the toolbar of the Execute SQL window.

   b. Select the `Access Create Generic Recordset Result Tables.sql` file located in the `<TestStand>\Components\NI\Models\TestStandModels\Database` directory.

      After you select the file, the **SQL Commands** control contains the set of SQL commands used for creating the default result tables.

   c. Click **GO** on the toolbar to execute the SQL commands.

   d. Review the results of the SQL commands in the **SQL History** control of the Execute SQL window.

   e. Close the Execute SQL window.

5. Select **Window»Refresh** to verify that the tables were created successfully.

   The Data Link window should now contain a list of all the tables created in the database, as shown in Figure 10-14.

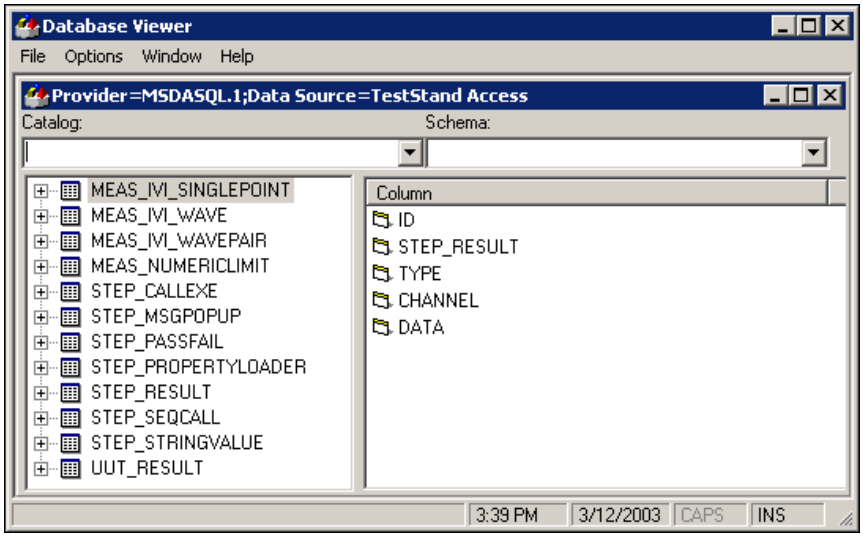


Figure 10-14. Data Link Window Displaying Tables

6. Close the Database Viewer application.
7. Click **OK** to close the Database Options dialog box.

End of Part C

Part D: Logging and Viewing Test Results

In this part of the exercise, you will run a computer motherboard test demo and log the results of the test to the database tables you just created. You will then view the test results stored in the tables.

1. Open the `Computer CVI.seq` sequence file located in the `C:\Exercises\TestStand I` directory.
2. Select **Execute»Test UUTs**.
3. Enter the serial number `A001` in the UUT Information dialog box. Although any serial number is valid, for the sake of conformity, enter the serial number `A001`.
4. Click **OK** in the UUT Information dialog box to launch the Test Simulator dialog box.
5. Select any component to fail and click **Done**.

   Additional Information
   TestStand launches a dialog box that indicates whether the test passed or failed. Click **OK** to close the dialog box. Notice the status text that appears at the bottom of the Execution window. TestStand notifies you when it is generating a report and logging UUT results to database. If you choose to test more than one UUT during an execution of a sequence file, the connection to the database remains open until the last UUT test is complete.

6. Repeat steps 3 through 5 for serial numbers `A002`, `A003`, and `A004`.
7. Click **Stop** in the UUT Information dialog box to end the execution. TestStand automatically generates and displays the final report.
8. Select **Tools»Database Viewer** to view the data that TestStand logged.
9. Before the viewer can display the data in the database you must specify a data link to the database.
   a. Select **File»New Data Link** to open the Data Link Properties dialog box.
   b. Click the **Connection** tab.
   c. Select **TestStand Access** from the **Use data source name** ring control.
   d. Click **OK** to close the Data Link Properties dialog box.
   e. The Database Viewer should now display the Data Link window containing the tables in the database as shown in Figure 10-15.

      Remember that the title of the Data Link window title resembles `Provider=MSDASQL.1;Data Source=TestStand Access`, which reflects the connection string of the data link.

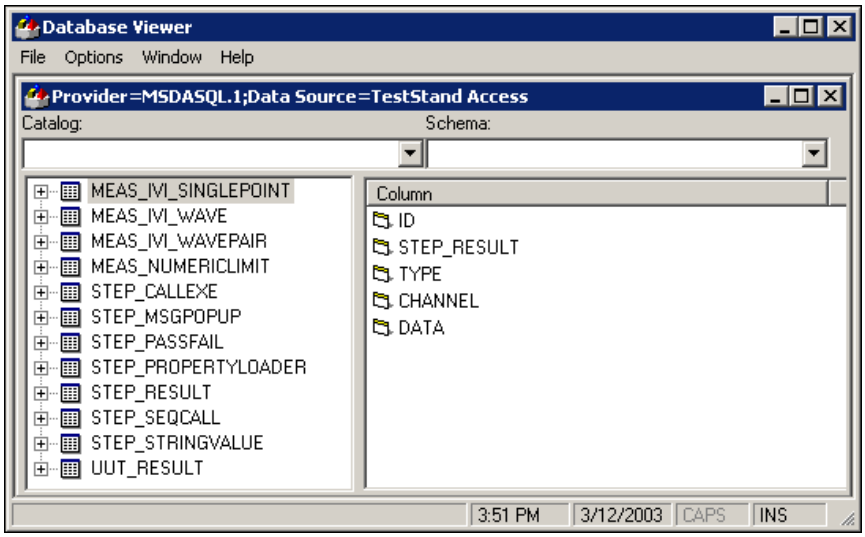


Figure 10-15. Data Link Window

10. In the Data Link window, you should see a list of tables including `UUT_RESULT` and `STEP_RESULT`. Right-click the `UUT_RESULT` table and select **ViewData** from the context menu.

    The Data View window, shown in Figure 10-16, contains the tabulated data as it would appear in Access. Notice that each UUT test has a unique ID. This ID is used for referencing step data in the `STEP_RESULT` table. Use the horizontal scroll bar to view data corresponding to each UUT test.

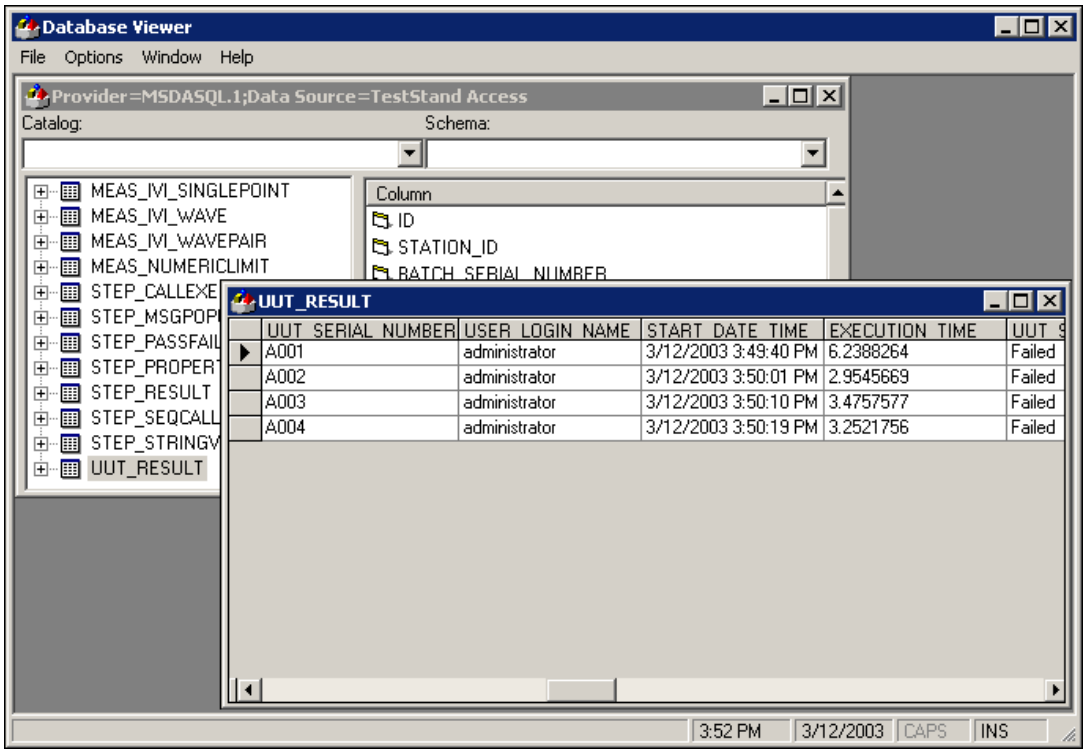


Figure 10-16. Data View Window Containing UUT Results

11. Return to the Data Link window by selecting it from the **Window** menu.
12. Right-click the `STEP_RESULT` table and select **ViewData**.

    The viewer launches a new Data View window, shown in Figure 10-17, that contains the test results for each step tested in the UUT test. You can also view the specific test result data for each step by viewing the data for its corresponding table in the database.

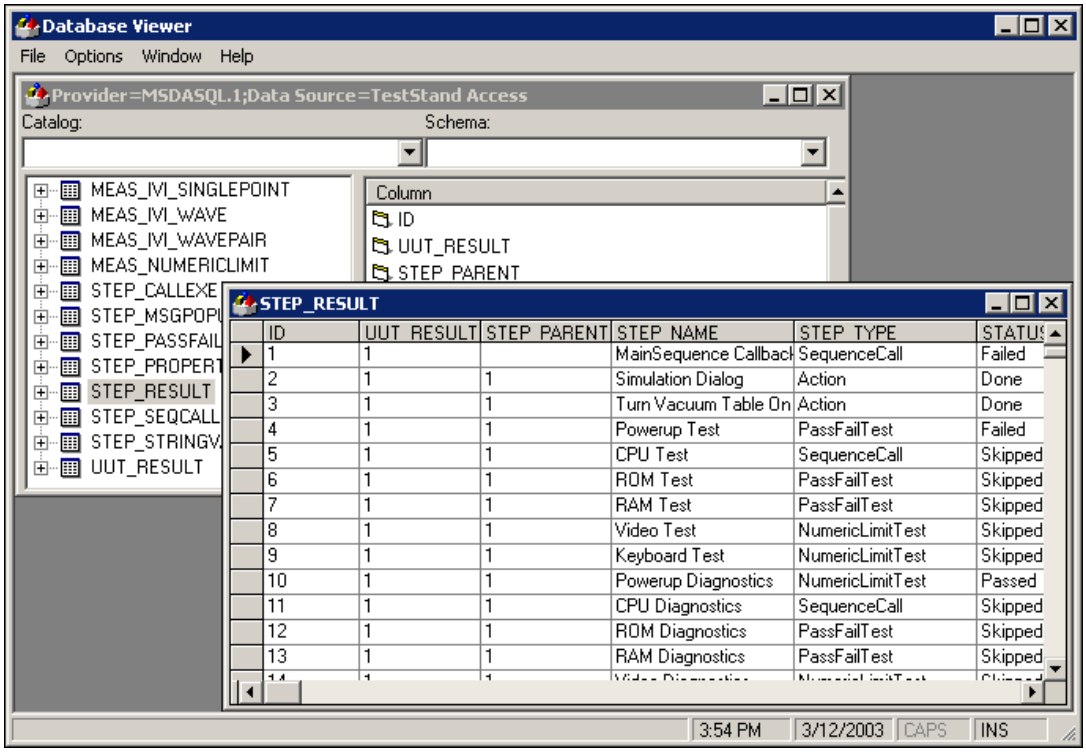


Figure 10-17. Data View Window Containing Step Results

13. Close the Database Viewer application.

End of Part D

End of Exercise 10-1

Self Review

1. What is the relationship between a record, table, database, and field?

2. If you want your results logged to the default Access Database, what do you need to do?

3. What is the Database Viewer, and what is it used for?

4. What are the three different windows used within the Database Viewer?

Notes

Notes

11

Custom Database Logging

Lesson 11: Custom Database Logging

In this lesson, you will:

- **Use SQL commands to communicate with databases.**
- **Use the database logging schemas to control how TestStand logs data.**
- **Use database step types to perform additional database operations**

Introduction

This lesson describes how to customize the database logging features in TestStand. The lesson covers how to modify the various components of database logging to meet your requirements.

Structured Query Language (SQL)

- SQL is a standard for database access
- Useful SQL commands:
  - CREATE TABLE
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

Structured Query Language (SQL)

SQL commands are supported by most major database management systems (DBMS). You can use SQL commands to manipulate the rows and columns in database tables. TestStand uses SQL commands exclusively to communicate with databases to create the required default tables, and to log UUT results and step results to a database table. TestStand includes the following five built-in SQL commands:

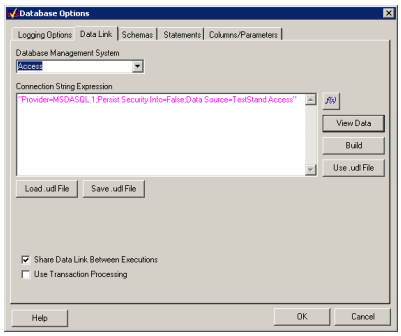
- **CREATE TABLE**—Creates a new table specifying the name and data type for each column.
- **SELECT**—Retrieves all rows in a table that match specific conditions.
- **INSERT**—Adds a new record to the table. You can then assign values for the columns.
- **UPDATE**—Changes values in specific columns for all rows that match specific conditions.
- **DELETE**—Deletes all rows that match specific conditions.

For a more detailed explanation of SQL commands, operators, and functions, refer to the *TestStand Help*. This version of SQL is included in the ODBC standard and applies to all ODBC-compliant databases.

Configuring Data Links

Select Configure»Database Options.

The Data Link tab is shown below.



Configuring Data Links

Use the Data Link tab in the Database Options dialog box to select which DBMS to use for logging and provide the required database connection information.

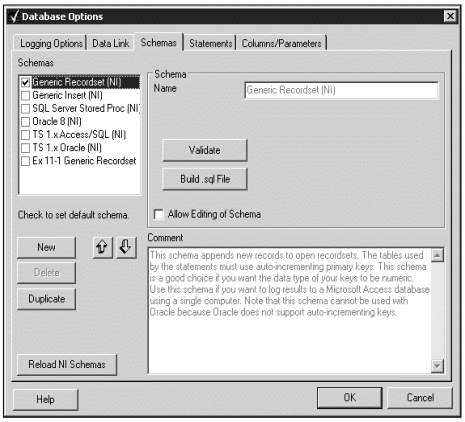
Use the Connection String Expression control to specify the connection string expression that TestStand uses to open a data source in which to log results. The string expression can be a literal value enclosed in quotations or an expression that is evaluated at run-time. Click **Expression Browser** to launch the Expression Browser dialog box and create the expression. Click **Build** to construct the connection string using the Data Link Properties dialog box. Refer to the *TestStand Help* for more information about the Data Link Properties Dialog Box.

An example connection string is shown above. This string can be stored in a Microsoft Data Link (`.udl`) file. You can use this file later by selecting the Find File button to point to the file. If you are using a Microsoft Data Link (`.udl`) file for the connection information, the Connection String Expression contains the file path and name of the file, for example, `"FILE NAME=C:\\Program Files\\Common Files\\System\\OLE DB\\Data Links\\Access.UDL"`

Configuring Schemas

Select **Configure»Database Options**.

The Schemas tab is shown below.



Configuring Schemas

Select the schema you wish to use from the Schemas tab in the Database Options dialog box. A schema consists of a list of statements which indicate to TestStand which tables the results should be logged to. The Statements tab contains the statements corresponding to the selected schema. Refer to the *Configuring Statements* section of this lesson for more information about the Statements tab. From the Schemas tab you can add and remove schemas, indicate whether the schemas can be modified, and change the priority of the schemas.

Modifying Database Schemas

- **You can modify database schemas to log data to your custom database tables**
- **Complete the following steps to modify the database schemas:**
  1. Create a copy of an existing schema.
  2. Select the copy.
  3. Enable editing of the schema.
  4. Edit Statements and Columns tabs.

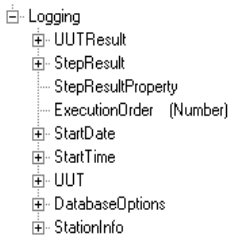
Modifying Database Schemas

The default schemas provided with TestStand for logging results to a database provide you with the basic tools you need to quickly set up an off-the-shelf test station that is preconfigured to log your test results to tables in a database. In some cases you might need to log your results to a pre-existing database that contains different tables than the default schemas. TestStand has the ability to modify or create new database schemas so that you can log the results to your custom database tables.

National Instruments recommends that you do not directly modify the existing schemas. Make a copy of the existing schema that most closely resembles your database logging needs then modify the copy of the schema. This way the default schemas are available if necessary. The steps shown above illustrate how to copy an existing schema so you can modify it to meet your database logging needs. Copy an existing schema using the **Duplicate** button. Make the new schema the active schema by placing a checkmark in the checkbox adjacent to the schema. Enable the **Allow Editing of Schema** option so you can edit the corresponding statements, columns, and/or parameters. The following slides discuss how to finish modifying your custom schema.

Logging Property

- **Copy of data or properties for logging:**
  - Database settings
  - Process model data structures
  - Current processed results
- **Used as preconditions and value expression during logging**



Logging Property

When the database logger starts, it creates a temporary property named Logging in the SequenceContext where it evaluates expressions. The Logging property contains subproperties that provide information about database settings, process model data structures, and the results that the logger processes. As logging progresses, it updates subproperties of Logging to refer to the step result or measurement the logger is processing.

In database logging, you can reference Logging subproperties in the precondition and value expressions. The references are entered into statements. Refer to the *Creating Custom Statements* section of this lesson for more information about statements.

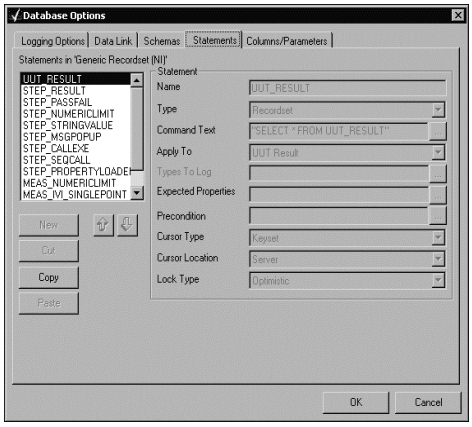
Use the parameters passed to the Log To Database sequence of the `<TestStand>\Components\NI\Models\TestStandModels\Database\Database.seq` to create the logging property

Refer to Chapter 6, *Database Logging and Report Generation*, of the *TestStand Reference Manual* for more information about Logging subproperties.

Configuring Statements

Select **Configure»Database Options**.

The Statements tab is shown below.

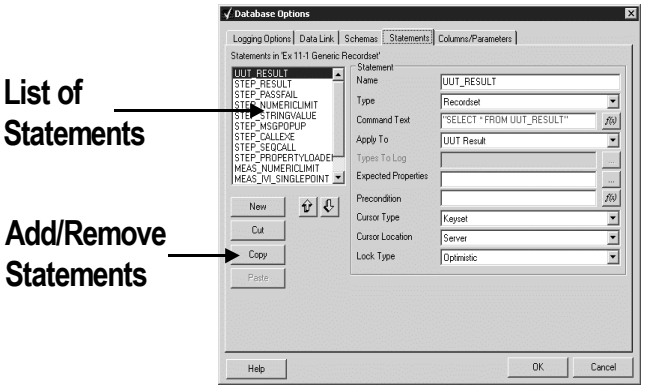


Configuring Statements

Statements define the type of results they operate on, the conditions that must be true before they can operate, and the database action to perform. In addition, statements can specify the data properties to log and the table columns in which to log. The list of statements can be found in the box on the left pane of the tab. After selecting a statement, the control settings of that statement display in the right pane of the tab. Refer to the *TestStand Help* for more information about the options on the Statements tab of the Database Options dialog box.

Creating Custom Statements

Specify your custom tables by modifying the contents of the Statements tab.



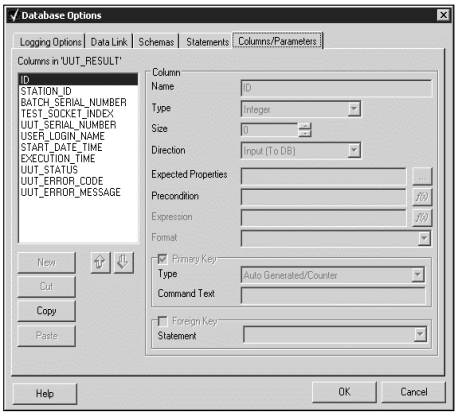
Creating Custom Statements

To continue creating your custom database schema, configure the Statements tab so that it accurately portrays the database tables you want to use. Use the Cut and New buttons to remove the statements that you do not want and add the custom statements that you want. For some of the elements you might be able to use the default settings. Refer to the *TestStand Help* for more information about the options on the Statements tab.

Configuring Columns

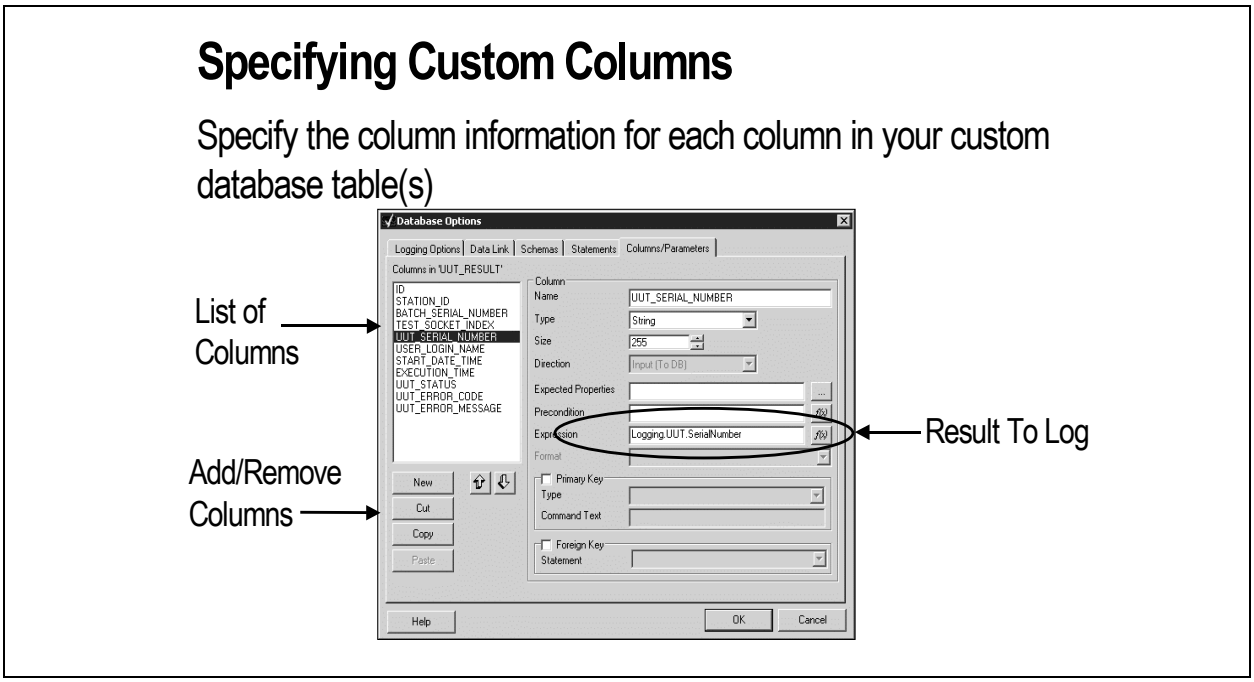
Select **Configure»Database Options**.

The Columns/Parameters tab is shown below.



Configuring Columns

Use the Columns tab of the Database Options dialog box for Record Set statements. Specify which columns the statement writes to in the new record. For each column, specify the data value to write.



Specifying Custom Columns

The last step in creating your custom database schema is to configure the **Columns/Parameters** tab for each table you are logging data to. Similar to the Statements tab, use the Cut and New buttons to remove the columns that you do not want and add the columns that you want. Refer to the *TestStand Help* for more information about the options on the Columns/Parameters tab.

Modifying Database Table Structures

Why?

- Useful when you want to modify what data is logged and there are no pre-existing tables with the required structure

How?

- Modify the scripts that create the tables

or

- Manually modify the default tables

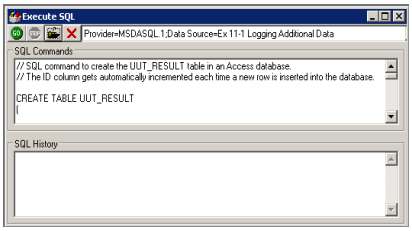
Modifying Database Table Structures

The Default tables and their fields might not be adequate for your logging needs. For example, you might have a new step type for which data needs to be logged to a new table, or you might have additional properties of a step that you want logged to a new field within an existing table. In these cases, you might need to modify existing tables or create new tables. There exists two methods for modifying the default table structure tools provided in TestStand.

- **Method 1**—Directly modify the script files using a text editor so when you use the script files in the Database Viewer to create the tables, TestStand automatically creates them with the structure you need. With this method, you modify the script files once and use them to create multiple databases that require that structure.
- **Method 2**—Use the Database Viewer to create the default set of tables using the script files (`*.sql`) that are provided. Once the tables are created, you can use the Database Viewer to manually add and remove columns from the tables to arrange your table structure. With this method, you manually modify the database each time you run the SQL script.

Creating Default Database Tables

- **SQL Script files for easily creating tables**
  - Creates required tables for specified schema
  - Scripts for Access, Oracle, and SQL Server
- **Use Database Viewer**
  - Select SQL script (*.sql)
  - Select GO button
  - Tables easily created



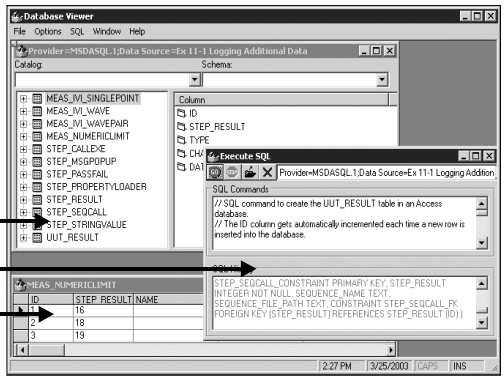
Creating Default Database Tables

TestStand includes the Database Viewer application for viewing data in a database table, editing table information, and executing SQL commands. You can use the Database Viewer to create the default result tables that the default schemas require. To use the Database Viewer application you must setup the DBMS server and any required DBMS client software.

TestStand comes with SQL script files for creating and deleting the default database tables that the schemas require. These script files are located in the `<TestStand>\Components\NI\Models\TestStandModels\Database` directory.

Database Viewer

- **Designed for:**
  - Executing SQL commands
  - Viewing data
  - Editing data
- **Views:**
  - Data Link
  - Execute SQL
  - Data View



Database Viewer

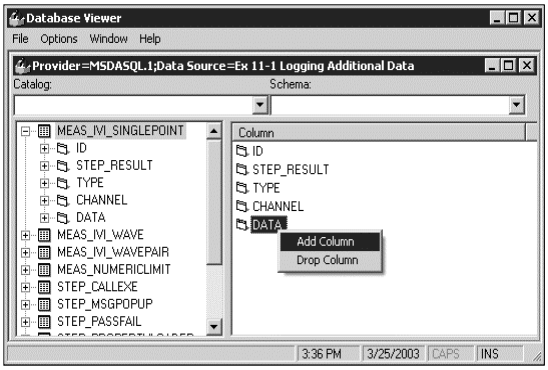
Use the Database Viewer to create, edit, or view database tables and data. Launch the Database Viewer by clicking the **View** button on the **Data Link** tab of the Database Options dialog box or selecting **Tools»Database Viewer**. The Database Viewer displays the following three types of windows:

- **Data Link window**—Contains a tree view of the tables that are defined for the data link. The list view displays the contents of the currently selected node in the tree view. You can display a context menu by right-clicking in the window. The items in the context menu vary depending on whether you right-click a node in the tree view, and entry in the list view, or the background area The context menu can contain the following items:
  - **View Data**—Opens a new Data View window with the contents of the table.
  - **Add Table**—Creates a new table in the DBMS.
  - **Drop Table**—Deletes a table from the DBMS.
  - **Add Column**—Adds a new column to a table.
  - **Drop Column**—Deletes a column from a table.
- **Execute SQL window**—Contains a SQL Commands control and a SQL History control. You can enter SQL commands in the SQL Commands control and execute its contents by clicking the **Go** button. You also can load SQL script files by clicking the **Open File** button. Use the **Clear** button to delete the contents from the SQL Commands control.

- **Data View Window**—Contains a grid display of the data returned from a SQL command. The Database Viewer automatically opens a new Data View window when you select **View Data** from the context menu or when you issue a SQL command that returns a recordset. You also can edit data in a table using this window.

Modifying Table Structures Manually

Use the Database Viewer to modify the default table structure to meet your needs



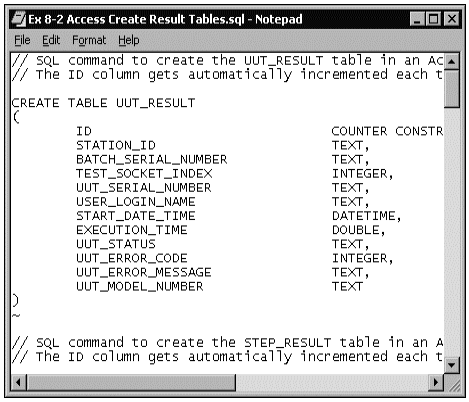
Modifying Table Structures Manually

This method assumes that you have already created the tables. The above slide illustrates how to use the Database Viewer to manually modify the structure of tables. This method might work well when you only want to make minor changes to the default table structure. If you want to make significant changes to the table structure, consider using the modify script method.

Note You can only add a column to the end of the column list using this method. If you want to insert a new column between two existing columns, first drop the columns below the place where you want to insert the new column, then add the dropped columns back individually. Columns in the Database Viewer are not necessarily listed in the order in which they appear in the table.

Modifying Script Files

- Edit SQL script files with a text editor



- Use the Build .sql File button on the Schemas tab

Modifying Script Files

This method involves copying and modifying a default script file using a text editor. This gives you the ability to easily create the tables with your custom table structure by calling the script files from the Database Viewer. This method allows you to create tables and fields in any order you want. With this method you modify a script file once, then use the modified script file to create additional tables with that structure on other computers using TestStand.

Note TestStand can automatically create a SQL script to match a schema you create in TestStand. On the Schemas tab of the Database Options dialog box, click the **Build .sql File** button. TestStand builds the SQL file for the currently selected schema. Make sure you add all statements and columns/parameters to your schema before creating the SQL script.

Logging Additional Data to Databases

1. **Identify additional data to log.**
2. **Pass data with the parameters of the `Log To Database` sequence in the process model or add to step results container.**
3. **Determine how and where to store the data in your database.**
   a) Number of columns/tables to create
   b) Data types to use in columns
4. **Modify database schema and your database accordingly.**

Logging Additional Data to Databases

Using the default database schemas, TestStand logs all the test results to the database based on your database configuration. You also can modify the database schemas and your database using techniques to log additional data.

Complete the following steps to log additional data to the database:

1. Identify additional data that you want to log.
2. Pass process model data with the parameters of Log to Database sequence. Open the Log To Database sequence located in the `<TestStand>\Components\NI\Models\TestStandModels\Database\Database.seq` directory.

Note The parameters passed to the sequence include your test results, UUT and Station data, database options, and start date and time. TestStand uses these parameters to create the logging object from which data is logged to your database. If additional data is to be logged, it must be within the parameters passed to the Log To Database sequence.

The most common addition is the new step result data of UUT data. New step results are automatically passed with the other step results using techniques covered in the report customization section of this manual. UUT data must be passed from your process model entry point as well as the UUT serial number.

3. Determine how and where the additional data should be stored in the database. Consider how many columns you need to add and what data types each column requires to accommodate the additional data.
4. Modify database schema and table(s) to enable the additional data to be logged.

Exercise 11-1: Logging Additional Data to a Database

Objective: To modify the default Generic Recordset schema to allow for the logging of additional data to the database.

Estimated Time: 15 minutes

Refer to page 11-31 for instructions for this exercise.

Exercise 11-2: Modifying Database Tables to Log Additional Data

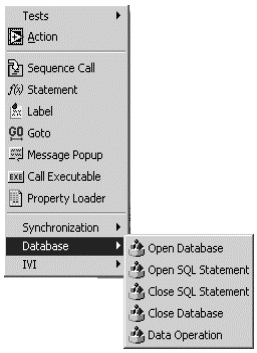
Objective: To modify the default database tables to allow for additional data to be logged to the database.

Estimated Time: 20 minutes

Refer to page 11-40 for instructions for this exercise.

Database Step Types

Primarily used from within test sequences because schemas handle database logging

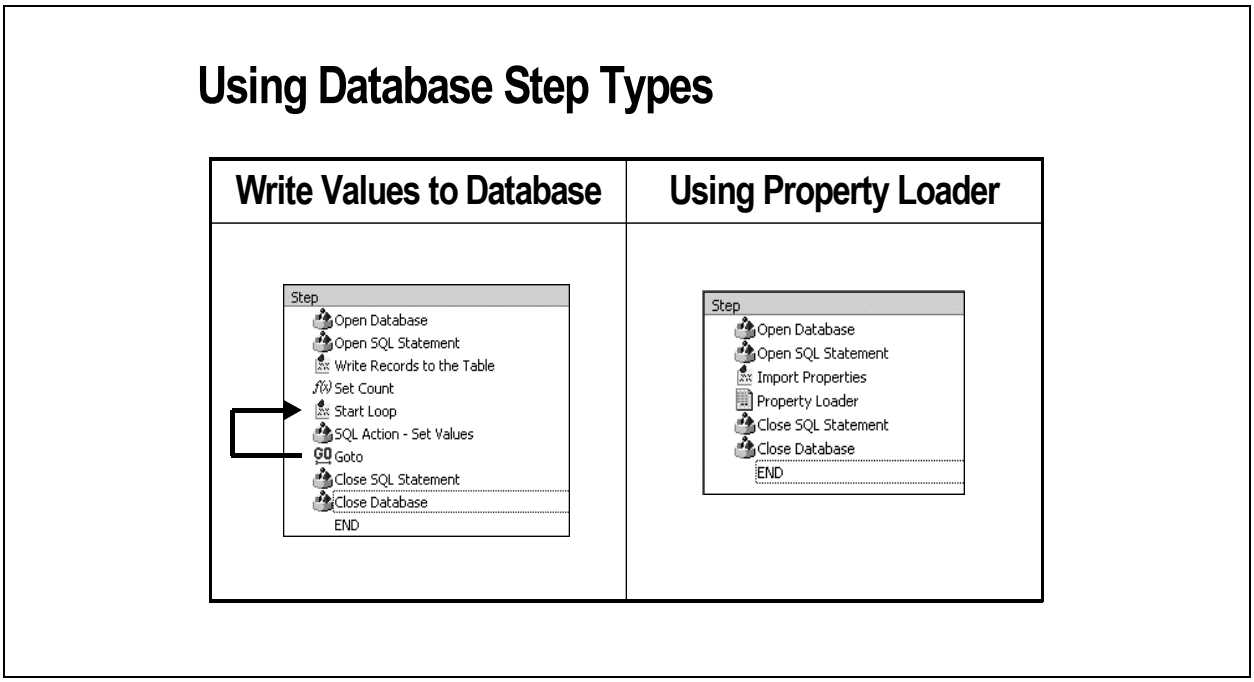


Database Step Types

When you use schemas, you need not use the Database steps for logging unless you want to communicate with the database from within the test sequence.

The five Database step types provide the following functionality:

- **Open Database**—Connects to a database.
- **Open SQL Statement**—Selects data within a database table.
- **Close SQL Statement**—Closes references to the table.
- **Close Database**—Disconnects from the database.
- **Data Operation**—Creates new records, deletes records, and gets and updates existing records.



Using Database Step Types

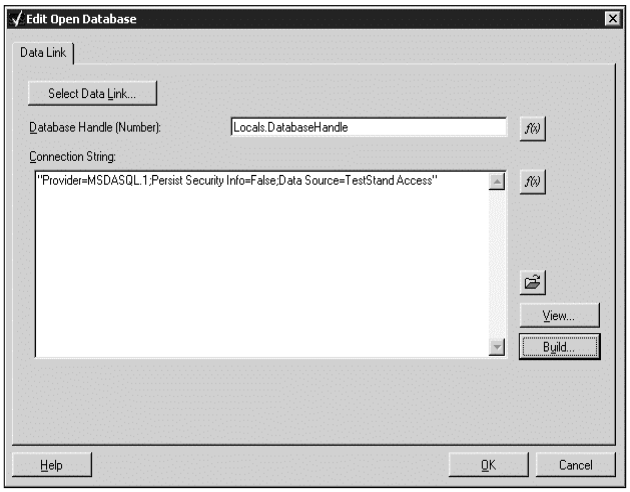
There are two common cases in which you would use the Database step types. The first case involves writing/reading data to a database from the test sequence. Complete the following steps to write/read data to a database from the test sequence:

1. Open a connection to the database using the Open Database step. This step returns a database handle (number) to be used in an Open SQL Statement step.
2. Select a set of records from a table on which to operate by issuing an SQL query within an Open SQL Statement step. This step returns a statement handle to the selected data, after which you can perform multiple operations on that data set using Data Operation step.
3. Read/write the data to the database using the Data Operation step type. If you are reading values from the database, use the Get operation. If you are writing values to your database, use the Set and Put operations. You may loop on these steps depending on the number of values you want to read/write.
4. After you finish interacting with the database, close both the statement and database handles that you opened using the Close SQL Statement and Close Database steps, respectively.

The second case involves using the property loader. This is the most common case of using the Database step types. When you use the property loader, you must open both an Open Database step and a SQL Statement

step as described previously. Remember that you must close these handles once you finish loading properties from the database.

Open Database Step Type

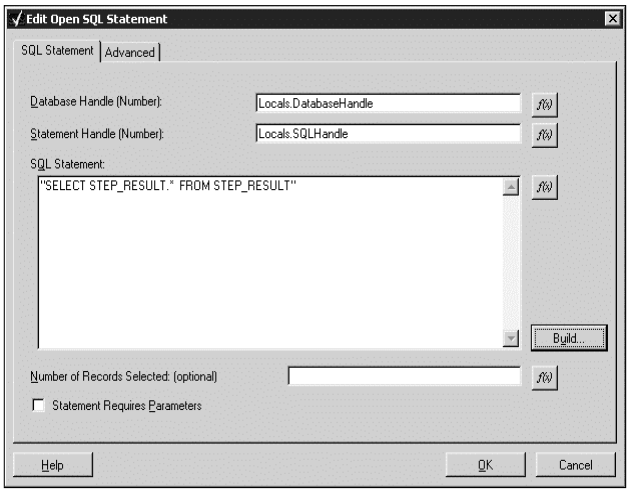


Open Database Step Type

The Open Database step type opens a database for use in TestStand. An Open Database step returns a database handle (a number) that you can use in an Open SQL Statement step.

The Data Link dialog box is similar to the Data Link tab of the Database Options dialog box. The only difference is the Database Handle control, which you use to specify the name of a variable or property of type Number to which you assign the returned database handle.

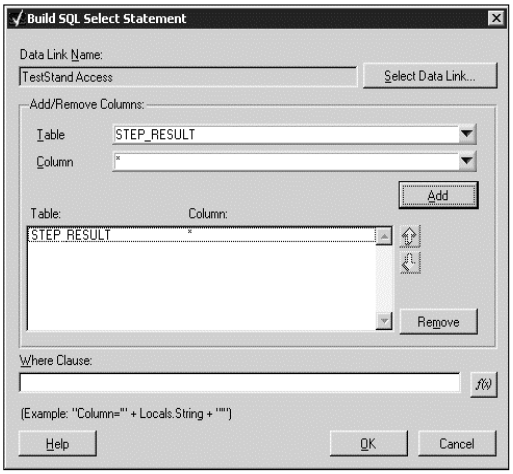
Open SQL Statement Step Type



Open SQL Statement Step Type

This step requires a database handle obtained from a preceding Open Database step. The Open SQL Statement Step issues a SQL command that is usually a SELECT command which returns a handle to records you select from a database table. Clicking the **Build** button launches the Build SQL Select Statement dialog box, which you use to construct a SQL SELECT statement. For more information on the SQL SELECT command and the options contained in the Build SQL Select Statement dialog box, refer to the *TestStand Help*.

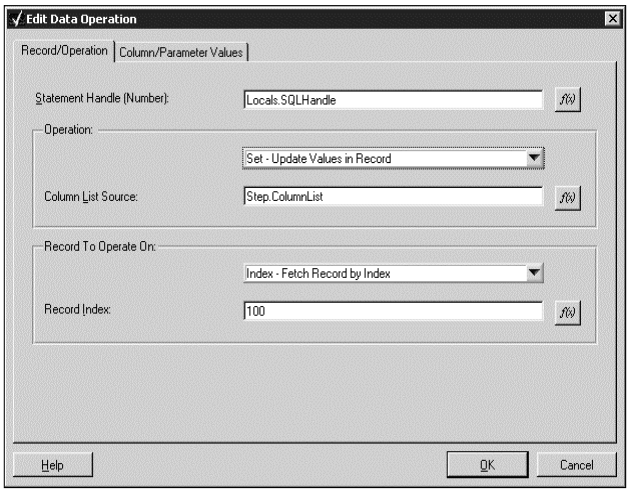
Open SQL Statement: Building an SQL Select Statement Dialog Box



Open SQL Statement: Building an SQL Select Statement Dialog Box

You can use the SELECT, CREATE TABLE, DELETE, DROP TABLE, and INSERT SQL commands with the Edit Open SQL Statement. The most common SQL command used in the Edit Open SQL Select Statement is SELECT, which returns a handle to a set of selected records. You can use the Build SQL Select Statement dialog box to construct an SQL SELECT statement expression. The Build SQL Select Statement dialog box also includes a Where Clause that is a literal string or expression used to search the database for a specific entry or entries.

Data Operation Step Type



Data Operation Step Type

The Data Operation step performs operations on the recordset returned by a preceding Open SQL Statement step. With the Data Operation step you can fetch new records, retrieve values from a record, modify existing records, create new records, and delete records.

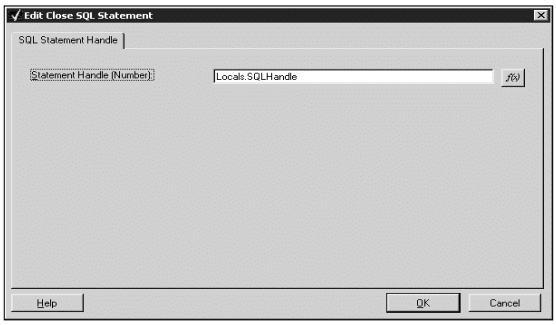
The Edit Data Operation dialog box contains the Record/Operation and Column/Parameter Values tabs. The Record/Operation tab allows you to specify a reference to selected records on which you wish to operate, how to reference these records while performing operations, and what operations to perform.

If the operation includes get or set, the Column/Parameter Values tab provides a mapping of TestStand properties and variables to the database field. That is, in which TestStand property the values from a database field should be stored, and vice versa. After you select the statement with which the set of records were selected, the Column Name/Number control is automatically populated. You can then select columns and enter the corresponding TestStand property, variable, or literal value to map to the column.

Refer to the *TestStand Help* for more information about the options in the Edit Data Operation dialog box.

Close SQL Statement Step Type

Closes an SQL statement handle that you obtain from an Open SQL Statement step



Close SQL Statement Step Type

The Edit Close SQL Statement dialog box contains the Statement Handle option, which specifies the name of the variable or property of type Numeric that contains the statement handle to close. After closing the SQL statement handle, the step assigns a value of zero to the variable or property.

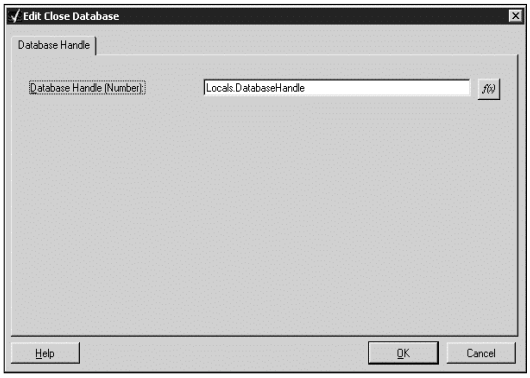
The Close SQL Statement step type defines the following step property in addition to the common custom properties.

- `Step.StatementHandle`—Specifies the name of the variable or property of type Number that contains the SQL statement handle to close.

Note TestStand does not automatically close SQL statement handles. You must call a Close SQL Statement to close open handles when you finish with them. If you abort an execution, you must exit the application process that loaded the TestStand Engine to ensure that TestStand frees all database handles. Selecting Unload All Modules does not close the handles.

Close Database Step Type

Closes the database handle that you obtain from an Open Database step



Close Database Step Type

The Edit Close Database dialog box contains the Database Handle options, which specifies the name of the variable or property of type Number that contains the database handle to close. After closing the database handle, the step assigns a value of zero to the variable or property.

The Edit Close Database step type defines the following step property in addition to the common custom properties.

- `Step.DatabaseHandle`—Specifies the name of the variable or property of type `Number` that contains the open database handle to close.

Note TestStand does not automatically close open database handles. You must call a Close Database step to close open handles when you finish with them. If you abort an execution, you must exit the application process that loaded the TestStand Engine to ensure that TestStand frees all database handles. Selecting Unload All Modules does not close the handles.

Lesson 11: Summary

- Modify the step Result List or the parameters of the Log To Database sequence to collect additional data for database logging
- Modify Database Logging Schemas to support custom databases or add additional data to default tables
- Use the database step types to directly communicate with a database from a sequence

Summary

This lesson described customizing database logging in TestStand. In order to modify the database logging procedure in TestStand, you must modify the database logging schema and the default table structure if there is not a pre-existing database that you are planning to use. You can collect additional data to log to a database by passing the additional data as a parameter to the Log to Database sequence and by modifying the corresponding schema and tables. The database step types allow you to directly interact with a database from within a sequence.

Exercise 11-1 Logging Additional Data to a Database

Objective: To modify the default Generic Recordset schema to allow for the logging of additional data to the database.

In this exercise, you modify the default database options and tables so you can log additional data to a database. This is useful when you have added additional properties to a step and want TestStand to log the values of those properties to a database along with the default properties that TestStand logs automatically.

Part A: Identifying Additional Data to Log

In this part of the exercise you will identify the two additional properties to log to the database.

1. Open the `Ex 11-1 Logging Additional Data to Database.seq` sequence file located in the `C:\Exercises\TestStand I` directory.
2. Right-click the `Waveform Limit Test` step and select **Open Tree View** from the context menu. Select the **Main»Waveform Limit Test» Result»RMSValue** property as shown in Figure 11-1.

   The RMSValue step property stores the RMS value calculated from the waveform generated by the code.

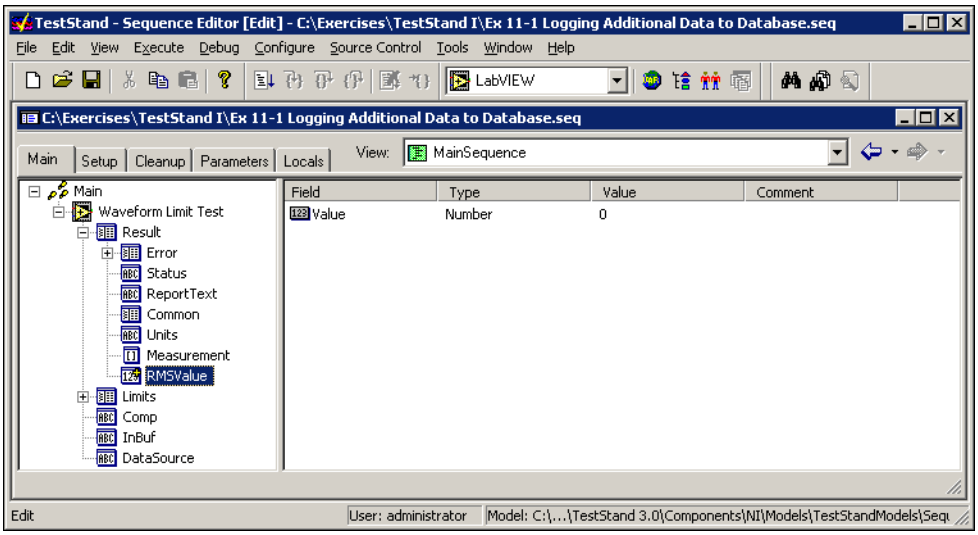


Figure 11-1. Tree View of Waveform Limit Test

Because the RMSValue step property is not a default property in the TestStand environment, TestStand does not automatically log it to a database. The default properties that are logged to a database are obtained from the ResultList property; therefore, you must verify that

the RMSValue step property is added to the ResultList after the step has completed executing.

There are several ways to add a property to the ResultList. In this case, TestStand automatically adds the RMSValue to the ResultList because it is a subproperty of the Step.Result property. All subproperties of the Step.Result property are automatically added to the ResultList.

End of Part A

Part B: Configuring Database Options

In this part of the exercise, you configure the database options to log additional data values to the database.

1. In the sequence editor, select **Configure»Database Options** to launch the Database Options dialog box.
2. Click the **Logging Options** tab and disable the **Disable Database Logging** option. Configure the other options as shown in Figure 11-2.

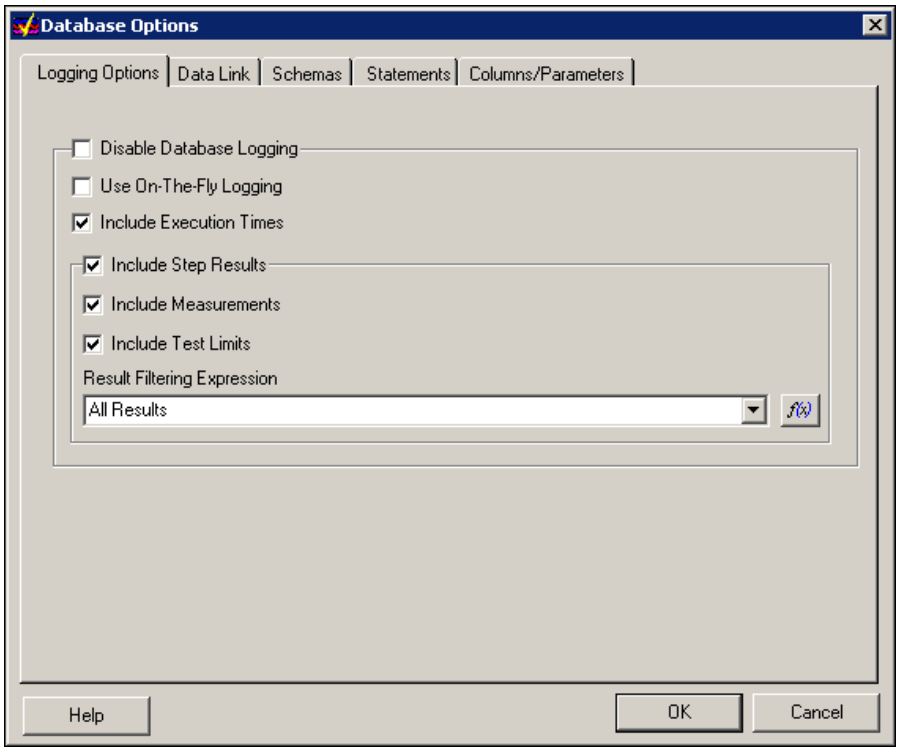


Figure 11-2. Database Logging Settings

3. Click the **Data Link** tab. Select **Access** for the **Database Management System** option, then click **Build** to launch the Data Link Properties dialog box.
4. Click the **Provider** tab, select the **Microsoft Jet 4.0 OLE DB Provider** option, and click **Next** as shown in Figure 11-3.

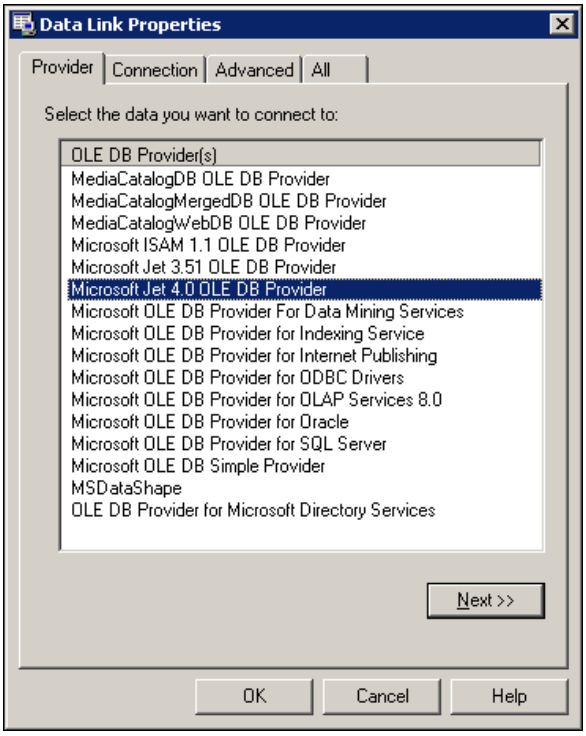


Figure 11-3. Data Link Properties Provider Tab

5. In the **Connection** tab, navigate to `C:\Exercises\TestStand I\Ex 11-1 Logging Additional Data to Database.mdb` for the database name and click **Test Connection** to confirm that you have a valid connection. Figure 11-4 shows the completed Data Link Properties dialog box.

   A dialog box should launch to indicate that the test connection succeeded.

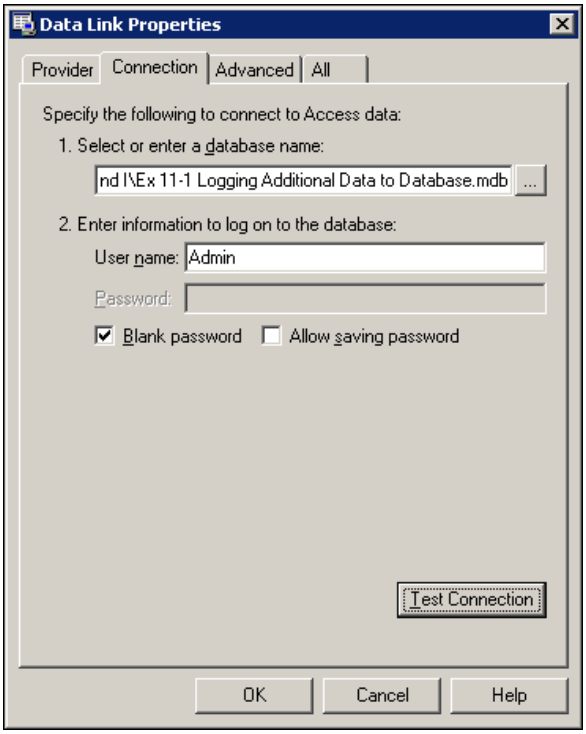


Figure 11-4. Data Link Properties Connection Tab

6. Click **OK** to close the Data Link Properties dialog box. Figure 11-5 shows the resulting **Data Link** tab settings on the Database Options dialog box.

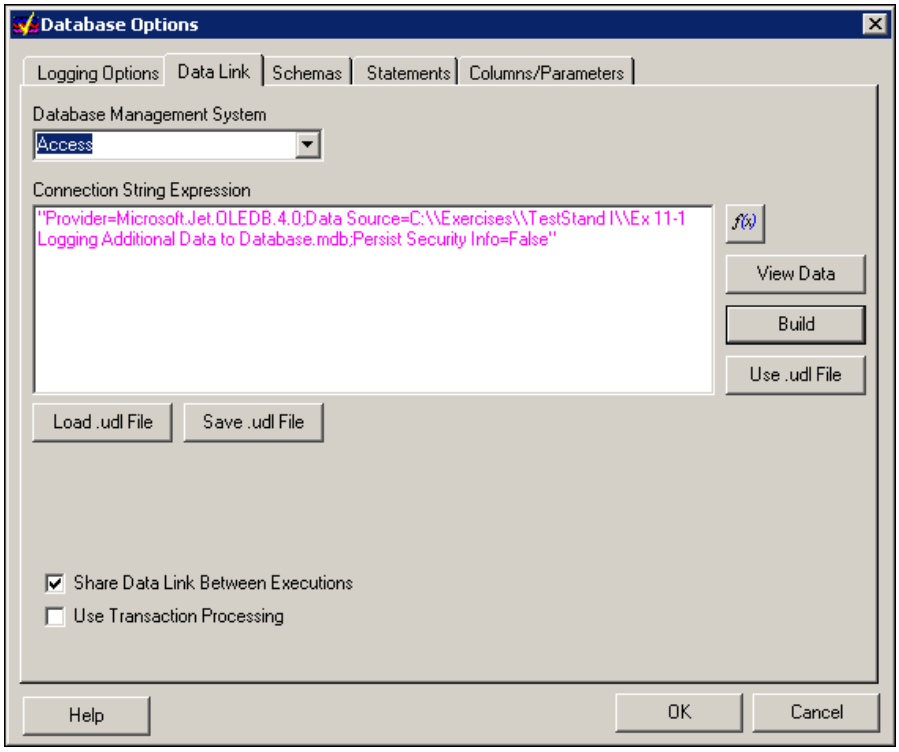


Figure 11-5. Database Options Dialog Box Data Link Tab

7. Click the **Schemas** tab and select the **Generic Recordset (NI)** schema. Click **Duplicate** to create a copy of this schema.
8. Select the new schema and enter `Ex 11-1 Generic Recordset` in the schema **Name** box.
9. Enable the **Allow Editing of Schema** option as shown in Figure 11-6.

   By creating a copy of the default Generic Recordset (NI) schema, you can make changes to the schema and always retain the default schema for later use.

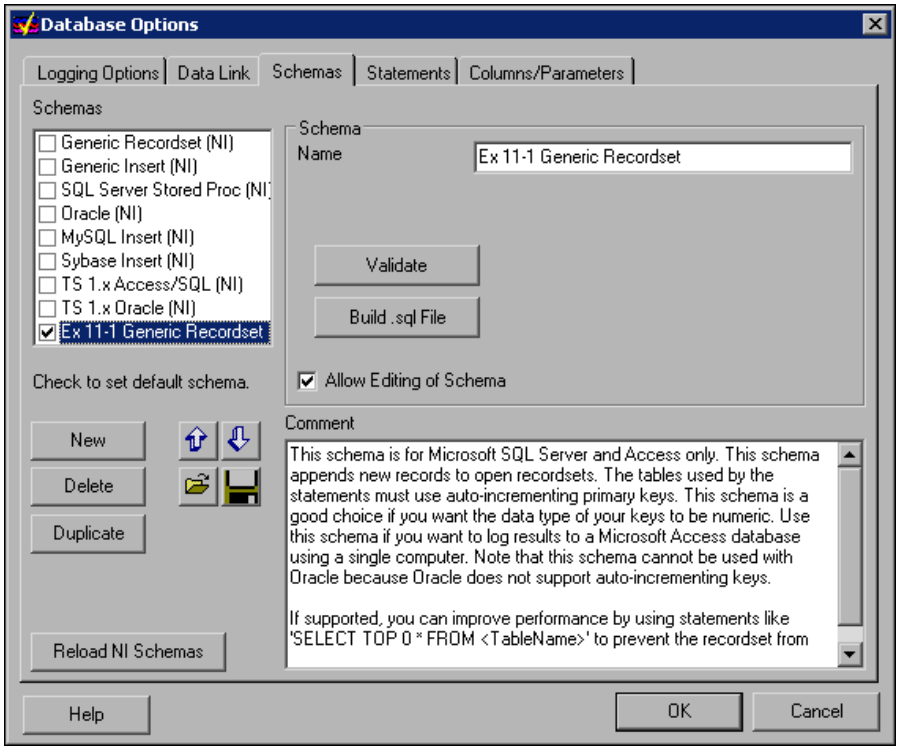


Figure 11-6. Database Options Schemas Tab

10. Click the **Statements** tab.

    Use the Statements tab to add statements for each additional data value you want to log to the database. You need to place the `RMS_VALUE` in the `STEP_NUMERICLIMIT` statement.

11. To add the `RMS_VALUE` to the `STEP_NUMERICLIMIT` statement, click the `STEP_NUMERICLIMIT` statement.

    For the schema to know to apply this statement to the `WaveformLimitTest` and `NumericLimitTest` steps, you must add the `Waveform_LimitTest` to the **Types To Log** control.

12. Click **Browse** next to the **Types To Log** control to open a dialog box that contains all the registered step types. By default, **NumericLimitTest** should be selected. Enable the **Waveform_LimitTest** option. Figure 11-7 shows the resulting Select Required Types For Statement dialog box.

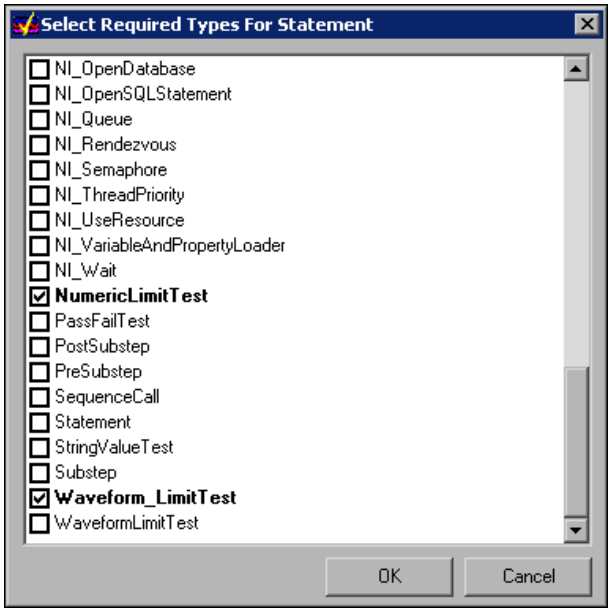


Figure 11-7. Select Required Types For Statement Dialog Box

13. Click **OK** to close the Select Required Types For Statement dialog box. Figure 11-8 shows the resulting Statements tab of the Database Options dialog box.

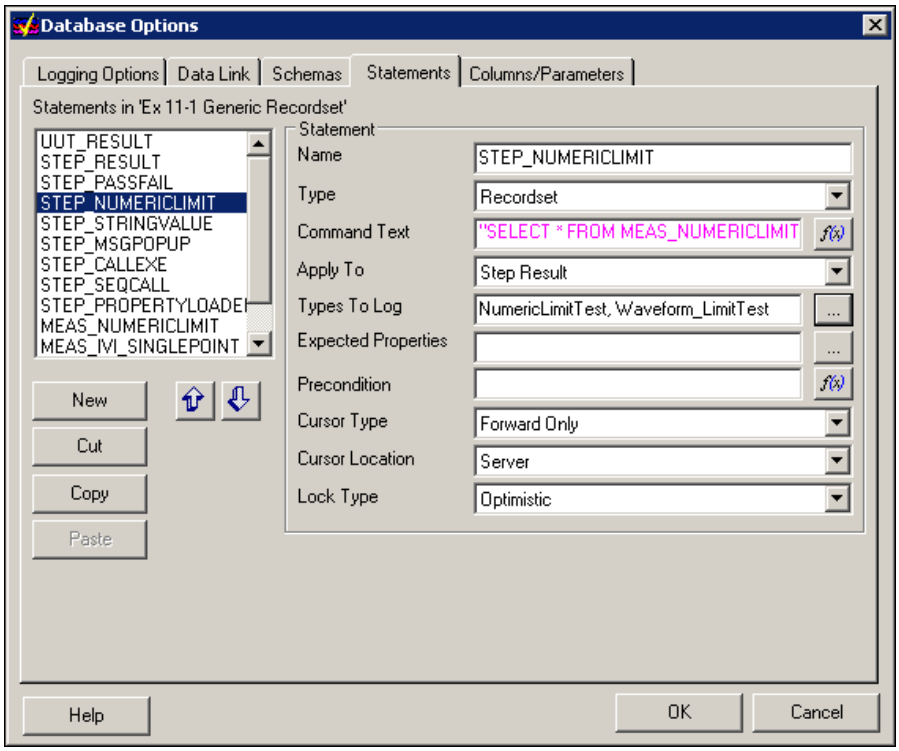


Figure 11-8. Database Options Dialog Box Statements Tab

14. Click the **Columns/Parameters** tab and select the `DATA` column. Click the **Copy** and **Paste** buttons to create a copy of the `DATA` column settings. Complete the parameters as follows.

Name: `RMS_VALUE`

Expected Properties: `Logging.StepResult.RMSValue`

Precondition: `PropertyExists("Logging.StepResult.RMSValue")`

Expression: `Logging.StepResult.RMSValue`

Figure 11-9 shows the resulting Columns/Parameters tab.

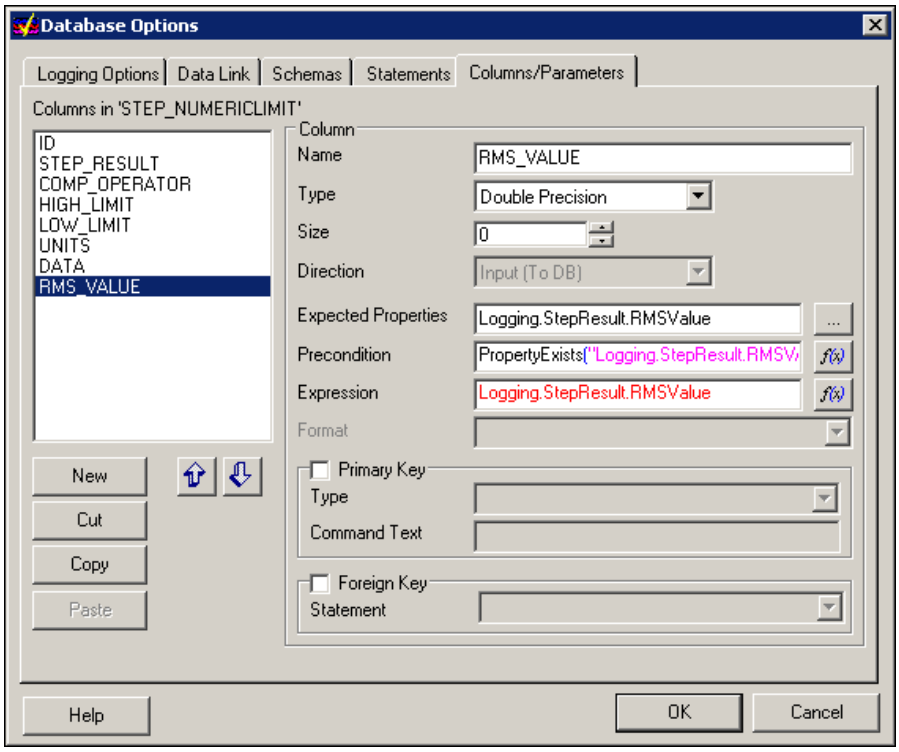


Figure 11-9. Database Options Dialog Box Columns/Parameters Tab

15. Select the **DATA** column. Copy the Expression field `Logging.StepResult.Numeric` to the Expected Properties field. Figure 11-10 shows the completed Columns/Parameters tab.

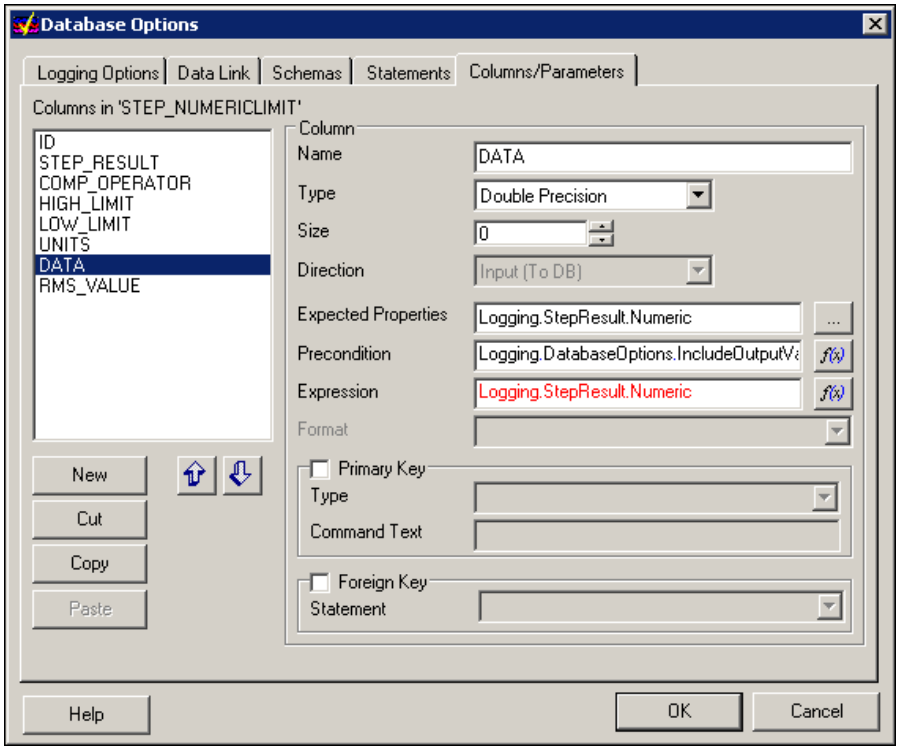


Figure 11-10. DATA Columns/Parameters Tab

Note You must add Logging.StepResult.Numeric to the Expected Properties of the Data column because two step types now use the STEP_NUMERICLIMIT statement. Without this expression, the database logger attempts to fill the data column when it logs a Waveform Limit Test step, thereby causing an error because the Step.Result.Numeric property does not exist for the Waveform Limit Test.

16. Click **OK** to store the changes and close the Database Options dialog box.

In the next exercise, you will create the actual database tables so the settings you configured in this exercise will correspond to the actual database columns and tables.

End of Part B

End of Exercise 11-1

Exercise 11-2 Modifying Database Tables to Log Additional Data

Objective: To modify the default database tables to allow for additional data to be logged to the database.

Now that you have configured the necessary database logging options in TestStand, you need to verify that your actual database tables are configured the same way. This exercise shows you how to modify the necessary database tables so that they correspond to the custom schema you created in Exercise 11-1.

Part A: Modifying the MEAS_NUMERICLIMIT tables

There are several different ways that you could modify the default table structures that TestStand uses to correspond to the custom schema you created in the previous exercise. In this exercise, you will directly modify one of the script files that is used to create the database tables.

1. In the sequence editor, select **Configure»Database Options** and click the **Schemas** tab.
2. Select the Ex 11-1 Generic Recordset schema and click **Build .sql File**.
3. A dialog box launches to prompt you to select the SQL file. Navigate to `C:\Exercises\TestStand I\`. Enter `Ex 11-1 SQL Script` in the filename field and click **OK**. TestStand creates the SQL script and opens the Database Viewer and and Execute SQL window. Figure 11-1 shows the Execute SQL window.

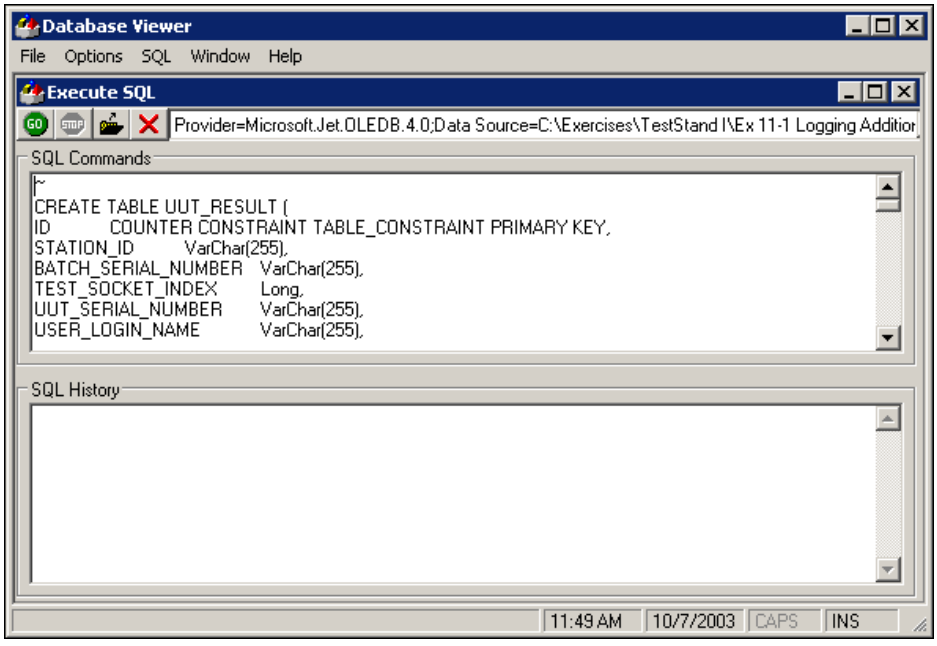


Figure 11-1. Execute SQL Window

4. Click **GO** on the toolbar to execute the SQL script that is loaded into the Execute SQL window. The executed commands should appear in the SQL History portion of the window as shown in Figure 11-2.

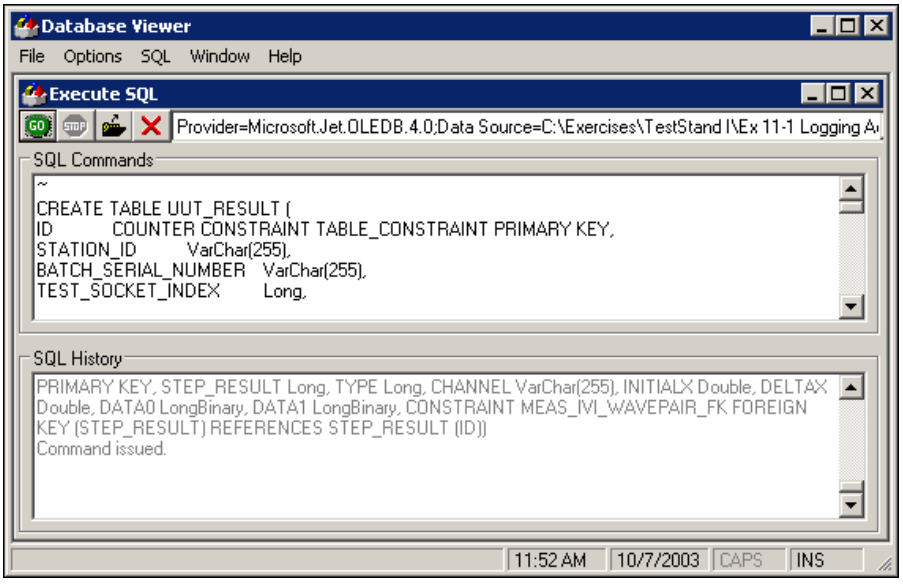


Figure 11-2. SQL History

5. Close the Execute SQL window. Select **Window»Refresh** from the Database Viewer application menu to show the newly created tables in the currently opened database. Verify that the `RMS_VALUE` column was added to the `MEAS_NUMERICLIMIT` table as shown in Figure 11-3.

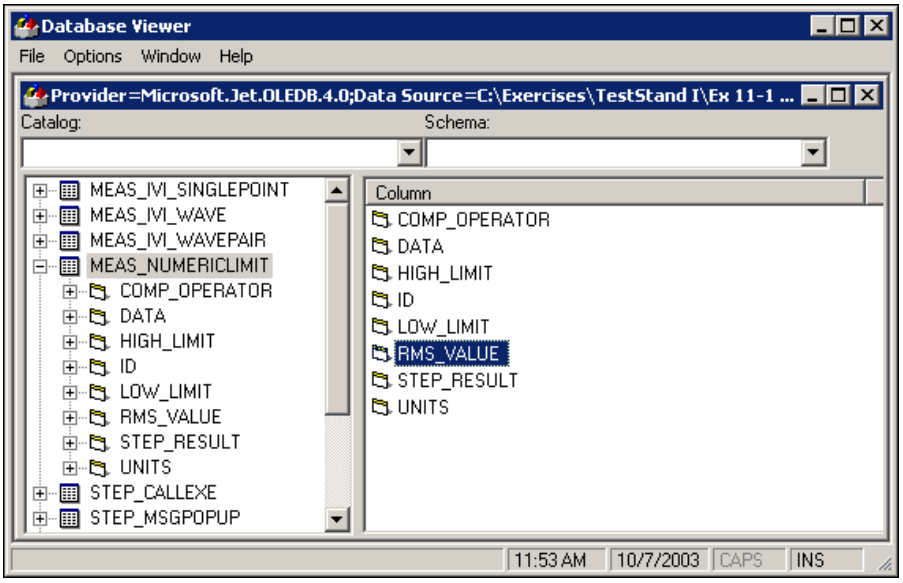


Figure 11-3. Updated Database Viewer Window

6. You have successfully created the required tables for the custom schema. Leave the Database Viewer application open for later use.

End of Part A

Part B: Running the Test Sequence

In this part of the exercise, you will execute the test sequence and see that the additional data was logged to the database.

1. Return to TestStand and click **OK** to exit the Database Options dialog box.
2. Open the `Ex 11-1 Logging Additional Data to Database.seq` sequence file located in the `C:\Exercises\TestStand I` directory if it is not already open.
3. Execute this test sequence using the Test UUTs execution entry point and test a few UUTs with varying sine wave amplitudes.
4. Return to the Database Viewer application, right-click the MEAS_NUMERICLIMIT table and select **ViewData** from the context menu.
5. Verify that the RMS_VALUES from the steps you have just executed are present. You can also view the data in the STEP_RESULT and UUT_RESULT tables to verify that the remainder of the data from the tests you performed has been logged correctly.
6. When you are finished, close the Database Viewer application and return to the sequence editor.
7. Return to TestStand and select **Configure»Database Options**.
8. Enable the **Disable Database Logging** option. You will not log results to the database for the remaining exercises.

End of Part B

End of Exercise 11-2

Self Review

1. What is the Logging property and when is it created?

2. When would you reference the subproperties of the Logging property?

3. What is the Data Link tab in the Database Options dialog box used for?

4. List the steps that must be taken to modify a database schema.

5. What are the two options for modifying database table structures?

6. What is a schema?

7. List the five Database step types and the action that each performs.

8. What are two common cases in which you would use the Database step types?

Notes

12

Distribution

Lesson 12: Distribution

In this lesson, you will:

- **Identify the various components of a TestStand system**
- **Use the TestStand Deployment Utility to distribute a test system to other computers**

Introduction

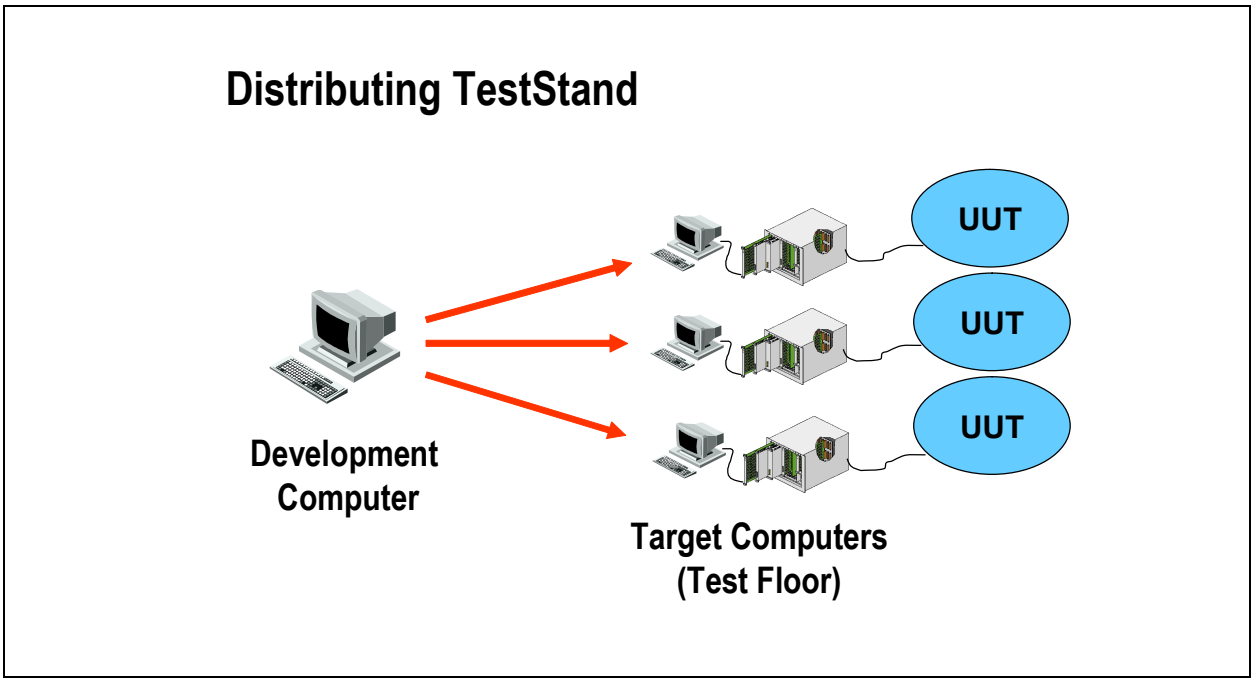
This lesson describes how to distribute your TestStand test environment once you are ready to deploy it.

TestStand System Components

- TestStand Engine
- Operator Interface
- Run-time Engines
- Process Models
- Step Types
- Configuration files
- Workspace Files
- Sequence Files
- Code Modules
- Hardware Drivers
- Supporting files
- Documentation

TestStand System Components

A TestStand system can rely on a wide variety of files and file types. In order to distribute a TestStand system from one computer to another, all the components of the TestStand system must be identified, collected, transferred to the target and deployed to the correct locations.



Distributing TestStand

A useful feature of TestStand is the ability to distribute a test system to other computers. The sequence editor helps develop test sequences but is not likely to be needed by an operator who will run the final test sequences. You can use the TestStand Deployment Utility to deploy TestStand to computers other than the development system.

TestStand Deployment Utility

- Tool for creating a custom TestStand distribution
- Launch from Tools»Deploy TestStand System
- Can create a deployable image or an installer
- Analyzes sequences to collect code modules
- Can include additional components in an installer
  - TestStand Engine
  - Operator Interface(s)
  - Hardware drivers
  - Run-time engines

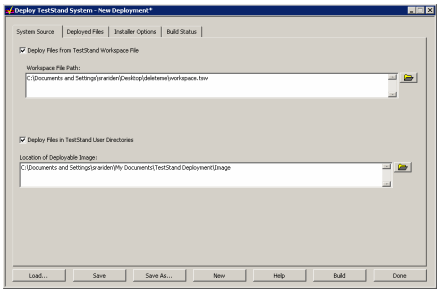
TestStand Deployment Utility

You can use the TestStand Engine to run TestStand sequences on computers other than the development system. The TestStand Sequence Editor provides a deployment utility for creating a deployable version of your TestStand test system. Select **Tools»Deploy TestStand System** to launch the Deploy TestStand System dialog box, which guides you through creating a Deployable TestStand installer.

The TestStand Deployment Utility simplifies the complex process of deploying a TestStand system by automating many of the steps involved in deployment, including collecting sequence files, code modules, and support files for your test system, and creating an installer for these files.

Using the TestStand Deployment Utility

- Identify components to deploy
- Determine whether to create an installer
- Create a system workspace file, if needed
- Configure and build the deployment



Using the TestStand Deployment Utility

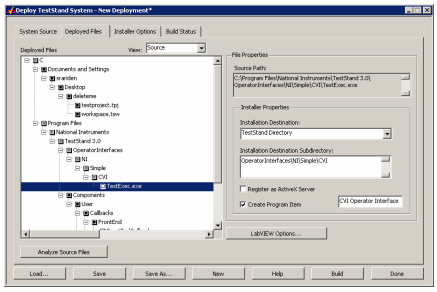
Use the following steps to deploy a TestStand test system using the TestStand Deployment Utility:

1. Identify the components to deploy.
2. You can choose to deploy files based upon a workspace file (`.tsw`) or files found in the `<TestStand>\...\User` directories.
3. Determine whether to create an installer.
4. An installer is necessary if you plan to deploy the TestStand Engine or any files found in the `<TestStand>\...\NI` directories.
5. Create a system workspace file, if needed.
6. Create a workspace file that contains all of the sequence files that your test system could execute. The deployment utility analyzes those sequence files to determine which files they reference, such as code module files. If you do not specify a workspace file, the installer includes only TestStand components and/or the contents of the user directory.
7. Configure and build the deployment.

   Use the deployment utility to configure which components are installed and the installer settings. Once you have created the deployment, take it to the target machine and run `setup.exe` to install your TestStand system.

Workspace Files

- Workspaces are the basis of deployment
- Add files to workspace
  - Sequence files
  - Operator interfaces
  - Additional files
- Each sequence in the workspace is analyzed to collect code modules and dependent files

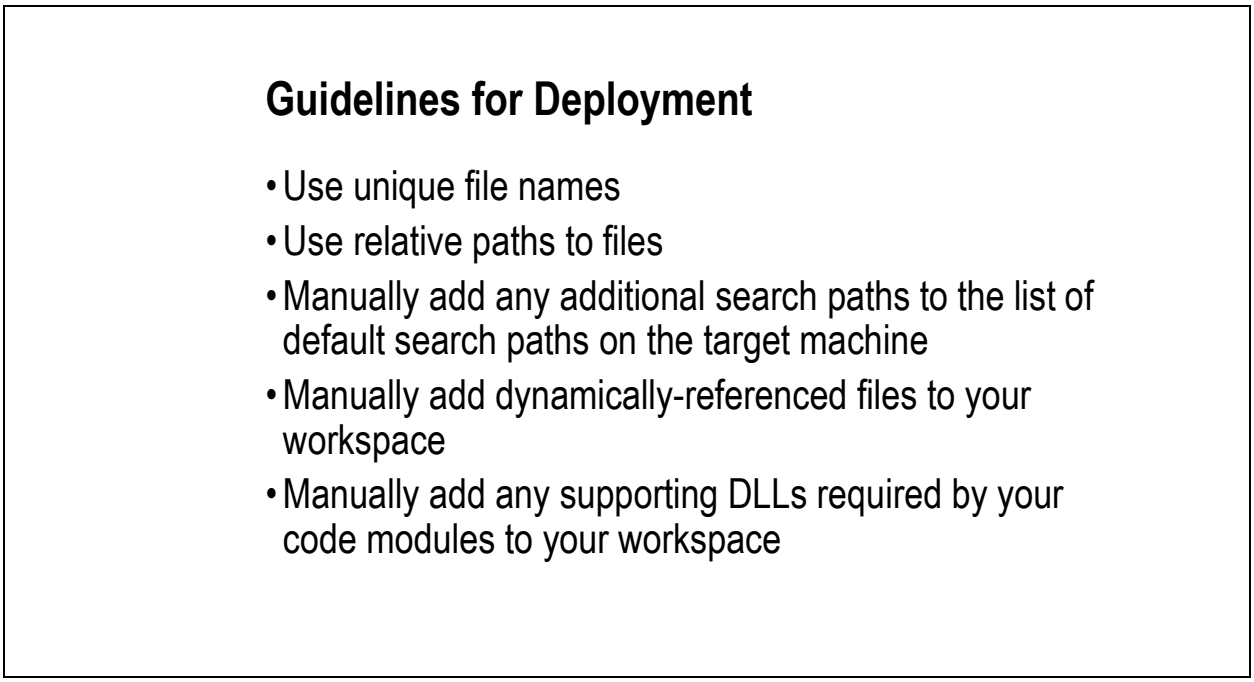


Workspace Files

The deployment utility uses workspace files to collect relevant files for distribution.

You should add sequences, code modules, and an operator interface to your workspace file. Using the workspace file can help you maintain an organized structure of your test system during development.

When deploying a workspace file, the deployment utility analyzes the workspace for any dependent files. For example, if your workspace contains a sequence file, the deployment utility searches the steps in every sequence of the file to find the referenced code modules. This analysis continues recursively until all files in the workspace hierarchy are analyzed.



Guidelines for Deployment

Use the following guidelines to create a successful deployment.

- Always use unique file names, even if you are working with a revision of an existing file. Ambiguous file names can cause the deployment utility to locate incorrect files, which can result in unexpected behavior.
- Use relative paths because relative paths allow TestStand to find files even if they were installed in a different location on the target computer than they were on the development computer. For example, you can locate a file that was saved in the `<TestStand>\Reports` directory on the development computer and in the `C:\TestStand\Reports` directory on the target computer using the relative path `Reports` because the TestStand installation directory is included in the default search path.
- Manually add any additional search paths to the list of default search paths on the target machine. The TestStand Deployment Utility will not copy additional search paths because the new directories may not exist on the target computer. Also, ambiguous file names in these search paths may cause TestStand to locate the wrong file.
- Manually add dynamically-referenced files to your workspace. Dynamically-referenced files include any sequences specified by an expression, property loader files specified by expressions, LabVIEW VIs called using VI Server, and dynamically-loaded DLLs.
- Manually add any supporting DLLs required by your code modules to your workspace. To avoid conflicts, do not add any DLLs that are part of TestStand or your operating system.

Distributing the Operator Interface

- Completely separate application from the sequence editor
- Build the operator interface into an EXE on a development machine
- Include the operator interface in your workspace file
- Add to deployment package using the TestStand Deployment Utility

Distributing the Operator Interface

You also need to distribute your operator interface to your target machine. Without the operator interface, you cannot view or run the sequence. To distribute your operator interface, build it into an executable and add the executable and any support files to your workspace file. When you use the TestStand Deployment Utility, it automatically includes the files that you added.

If you prefer not to distribute the operator interface with your TestStand deployment, you can use the same development environment you used to create the operator interface to make a distribution kit. Install that distribution kit on the target machine.

Note This is only required if you are using a custom operator interface. If you are using a default operator interface and you are installing the TestStand Engine, click **Engine Options** on the Installer Options tab and select one or more of the simple or full-featured operator interfaces.

Exercise 12-1: Deploying a Test System

Objective: Use the TestStand Deployment Utility to create an installer for a TestStand system.

Estimated Time: 15 minutes

Refer to page 12-11 for instructions for this exercise.

Lesson 12: Summary

- TestStand systems contain components other than a sequence file
- Use TestStand workspaces to organize and distribute test systems
- Use the TestStand Deployment Utility to distribute TestStand to target computers
- The *TestStand Reference Manual* contains additional information on deploying TestStand systems

Summary

Distribution is simplified by using the TestStand Deployment Utility. The deployment utility uses TestStand workspaces to make an executable which you can install on a target computer.

Refer to Chapter 14, *Deploying TestStand Systems*, of the *TestStand Reference Manual* for more information about deploying a TestStand system to one or more target computers

Exercise 12-1 Deploying a Test System

Objective: Use the TestStand Deployment Utility to create an installer for a test system.

In this exercise, you create an installer for a simple test system and use the installer to install the system.

Part A: Create a Workspace

In order to use the TestStand Deployment Utility, you must first create a TestStand Workspace and add a project to deploy.

1. Close any open windows in TestStand.
2. Select **File»New Workspace File** to create a new workspace.
3. Name the workspace `C:\Exercises\Ex 12-1 Deploying a Test System.tsw`.
4. Right-click in the empty file list and select **Insert New Project into Workspace** as shown in Figure 12-1.

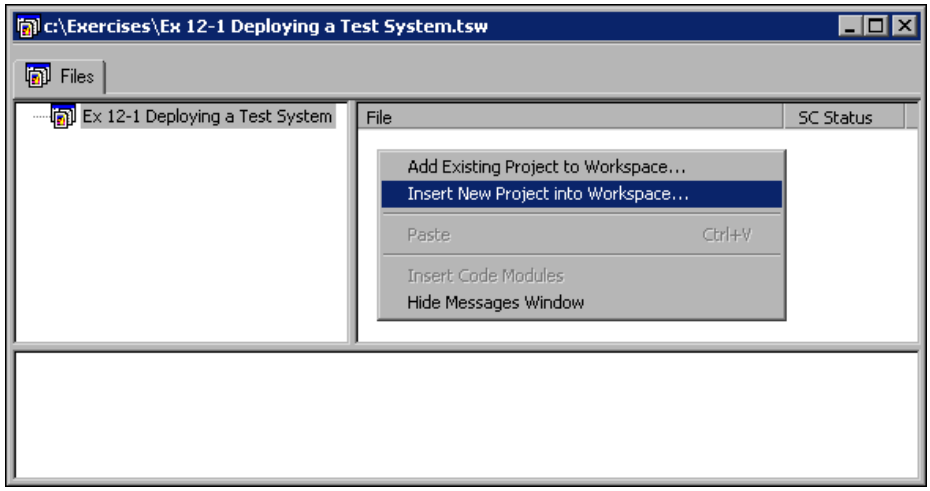


Figure 12-1. Inserting a New Project into a Workspace

5. Name the project `C:\Exercises\Ex 12-1 Deploying a Test System.tpj`.
6. Right-click the project you just created and select **Add Files to Project**.
7. Select `C:\Exercises\TestStand I\Ex 12-1 Deploying a Test System.seq` and click **Add**.
8. Click **OK** to exit the Add Files to Project dialog.
9. Double-click the sequence file you just added to open it. Run the sequence a few times and observe the behavior of the sequence.

Based upon the result of the Select Processor Message Popup, this sequence chooses to call one of three subsequences, each located in a different file.

10. Close the sequence.
11. Right-click **Ex 12-1 Deploying a Test System.seq** in the workspace window and select **Insert Code Modules**.

When you select **Insert Code Modules**, TestStand analyzes the sequence file and identifies any code modules or extra sequences which need to be added. Note that a project file can contain any other type of files, so you can manually add any additional files your project relies upon.

12. Observe that the three sequence files used by this sequence have been added to the project.

Note You can click in the inserted sequence files and select **Insert Code Modules** to insert the code modules from these sequences into the project. However, the deployment utility will analyze these sequences for you in Part B.

13. Save the Workspace

Tip If you are installing your test system to a computer without TestStand installed, you should include an operator interface in your workspace so that you can open and run your sequence files.

End of Part A

Part B: Creating an Installer

The TestStand Deployment Utility can be used to create an image or an installer for your test system. An image of your test system consists of a directory containing all the files used by your system that can be backed up or copied from one computer to another. When transferring an image of a test system, you are responsible for copying all the files to their correct locations on the target computer. Additionally, you must manually install any required components such as the TestStand engine or operator interfaces. Images are primarily useful when distributing simple test systems to computers which already have TestStand installed or when distributing test systems with a third party or custom installer.

When distributing complex test systems which contain customized components such as step types and process models or when distributing a test system to a computer which does not have the necessary drivers and engines for your system installed, you should use the TestStand Deployment Utility to create an installer for your system.

1. Select **Tools»Deploy TestStand System** to launch the TestStand Deployment Utility.
2. Check the **Deploy Files from TestStand Workspace File** checkbox.

Tip If you do not specify a workspace file, the deployment utility allows you to create an installer which will install only TestStand components, such as the Engine, Drivers, or custom user components.

3. Click **Browse** next to the **Workspace File Path** box to select `C:\Exercises\TestStand I\Ex 12-1 Deploying a Test System.tsw`.
4. If the workspace is not already open, click **Load Workspace in TestStand** to open it. Return to the deployment utility after the sequence file is loaded.

Tip If your test sequence utilizes any user-defined components such as custom process models or step types, check the **Deploy Files in TestStand User Directories** box to include user directories. The test system in this exercise does not use any user-defined components, so leave the box unchecked.

5. Create a directories to save the image and installer.
   a. Click **Browse** next to the **Location of Deployable Image** box and browse to the `C:\Exercises\TestStand I` directory. Do not click **OK**.
   b. Right-click in the directory and select **New»Folder**.
   c. Name the new folder `Ex 12-1 Distribution`.
   d. Open the `Ex 12-1 Distribution` folder you just created.
   e. Right-click in the directory and select **New»Folder**.
   f. Name the new folder `Image`.
   g. Right-click in the directory and select **New»Folder**.
   h. Name the new folder `Installer`.

You should now have a distribution directory for this exercise, and separate subdirectories to store the image and the installer.

   i. Open the `Image` folder and click **OK**.

Figure 12-2 shows the completed System Source tab.

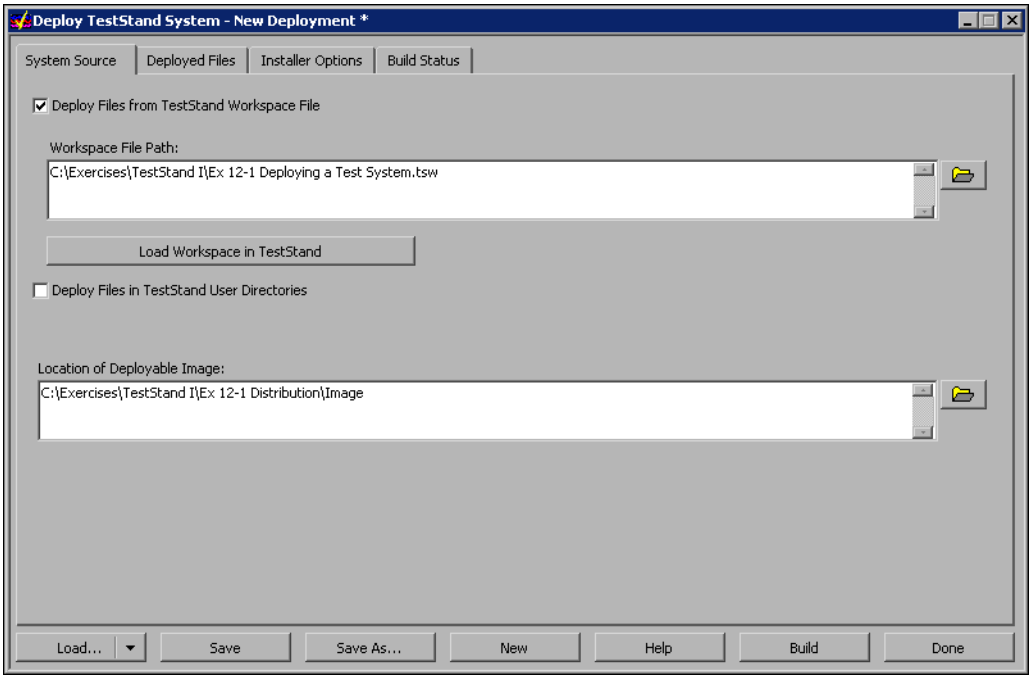


Figure 12-2. System Source tab of the TestStand Deployment Utility

6. Click the **Deployed Files** tab.
7. Click **Yes** when you are prompted to analyze the source files.

The deployment utility analyzes each sequence file in the workspace to identify any dependent sequence files or code modules. Note that the main sequence file and the three dependent sequence files that were included in the original workspace are now added to the distribution. In addition, the deployment utility analyzed the AMD Processor, Computer CVI and Intel Processor sequence files and determined that these sequences call a CVI code module, which is now included in the distribution.

Tip You can choose to remove the source code for DLL code modules and distribute only the DLL. Removing the checks next to the **Computer cvi.c** and **Computer cvi.prj** files will remove the source code for this exercise.

8. Highlight the **Ex 12-1 Deploying a Test Sytem.seq** file in the **Distributed Files** list to display the installation options for this file.
9. Check the **Create Program Item** check box and enter Ex 12-1 Computer Test in the adjacent text box.

Note The Create Program Item option instructs the installer to add a link to this item on the Windows Start Menu.

Figure 12-3 shows the completed Deployed Files tab.

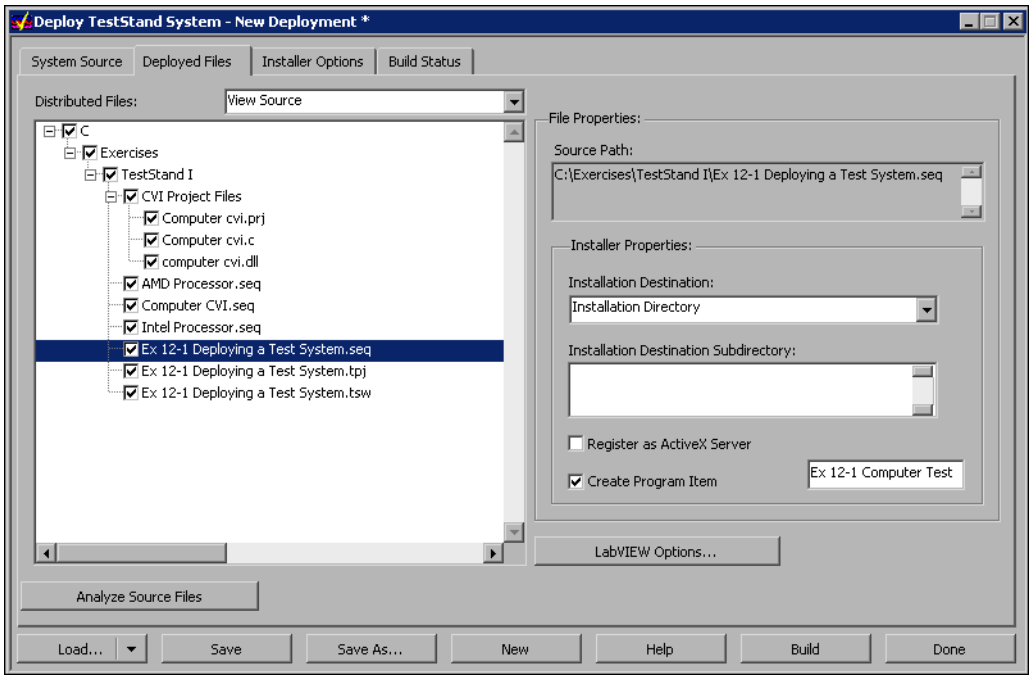


Figure 12-3. Deployed Files tab of the TestStand Deployment Utility

10. Click the **Installer Options** tab.
11. Ensure that the **Create Installer** check box is checked.

Tip If you do not check the Create Installer box, the deployment utility will only create a deployment image.

12. Change the **Installation Name** to `Ex 12-1 Computer Test System`.
13. Use the browse button next to the **Installer Directory** to select `C:\Exercises\TestStand I\Ex 12-1 Distribution\Installer`.
14. Change the **Start Menu Item Group** to `Ex 12-1 Computer Test System`.
15. Ensure that the **Default Installation Base Directory** is set to **Program Files Directory**.

Tip The Do Not Allow User to Change check box determines whether the installer will prompt the user to specify an installation directory. As a general rule, you should use relative paths in your test system so that it will function when installed to any directory. However, if you are forced to use absolute paths in some part of your test system, you can ensure that the test system is always installed to the default directory by using this check box.

16. Change the **Default Installation Subdirectory** to `Ex 12-1 Computer Test System`.

Note This will install the test system to the `C:\Program Files\Ex 12-1 Computer Test System` folder.

17. Click **Additional Components**.
18. Observe the additional components you can include in your installer and remove the check boxes next to any checked components.

Note For this exercise, you will be installing the test system to your current computer, and therefore all of the available components are already installed.

19. Click **OK** to close the Additional Components dialog.

Figure 12-4 shows the completed Installer Options tab.

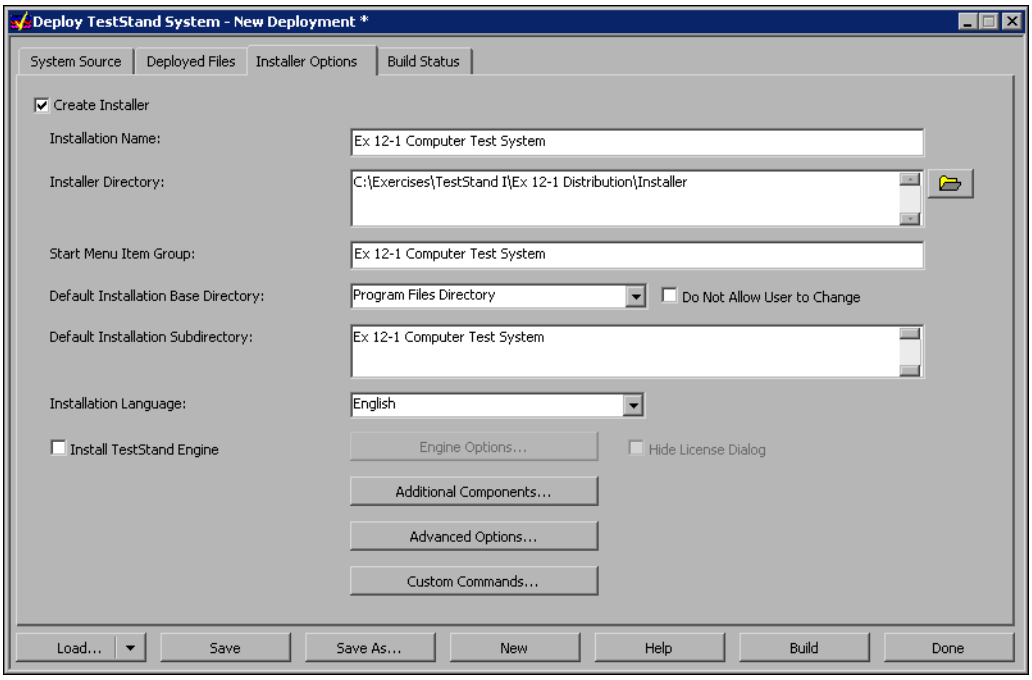


Figure 12-4. Installer Options tab of the TestStand Deployment Utility

Tip If you are installing your test system to a computer which does not have TestStand installed, you should check the **Install TestStand Engine** box to install the TestStand Engine on the target computer. For this exercise you will be installing the system to your current computer, which already has the TestStand Engine.

20. Click the **Build Status** tab.

21. Click **Save** and save the deployment as `C:\Exercises\TestStandI\Ex 12-1 Deploying a Test System.tsd`.

Note The deployment file contains the settings you have configured in the TestStand Deployment Utility. You do not need to save this file in order to build a distribution. However, saving the settings will prevent you from having to reconfigure the deployment if you need to change a setting or rebuild the distribution.

22. Click **Build** to create your distribution. Creating the distribution may take a couple of minutes.
23. Click **OK** when a dialog informs you that the build is finished.

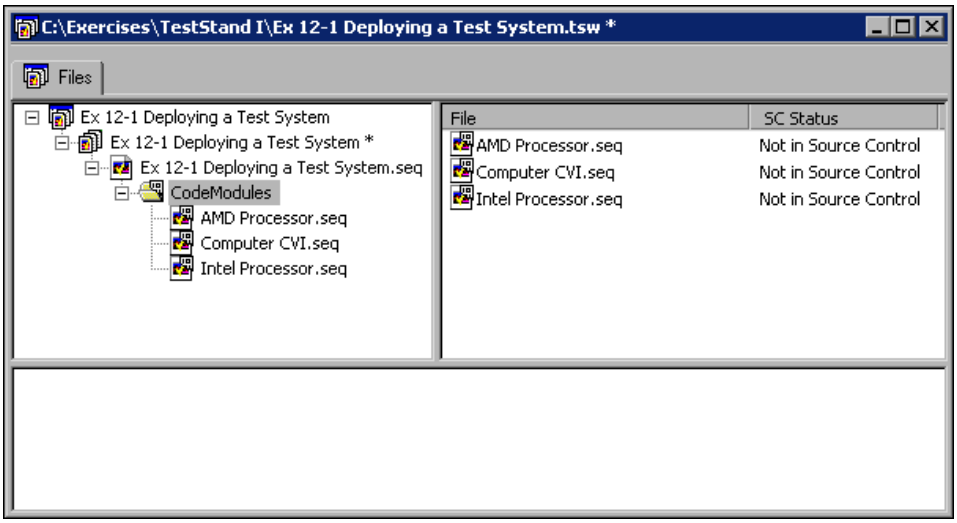


Figure 12-5. Completed Build

24. Click **Done** to exit the TestStand Deployment Utility.
25. Close TestStand.

Part C: Running the Installer

1. Open **Windows Explorer** and browse to the `C:\Exercises\TestStand I\Ex 12-1 Distribution` directory. Observe the directories you created for the image and the installer.
2. Open the `Image` folder and then open the `target` folder. This directory is an image which contains all of the files for your workspace.
3. Return to the `C:\Exercises\TestStand I\Ex 12-1 Distribution` directory.
4. Open the `Installer` folder and then the `Volume1` folder. This folder contains an installer for your test system which can be burned directly onto a CD.

5. Run the **Setup** program to begin the installation.
6. Leave the default installation directory and click **Next >>** twice to begin the installation.
7. When the installation completes, click **Finish**.
8. Using Windows Explorer, browse to the `C:\Program Files\Ex 12-1 Computer Test System` directory and ensure that the image for the test system is present.
9. Run **Start»Programs»Ex 12-1 Computer Test System»Ex 12-1 Computer Test** and ensure that the sequence file from this exercise opens.

Tip Because TestStand is installed on the computer you are using for this course, the sequence file opens in the Sequence Editor, which is the default program for opening a `.seq` file. If you are installing to a computer without TestStand, you should associate the `.seq` extension with an operator interface. The TestStand Deployment utility contains an option to do this in the Advanced Installer Options dialog.

End of Exercise 12-1

Notes

Notes

A

Introduction to IVI

Appendix A: Introduction to IVI

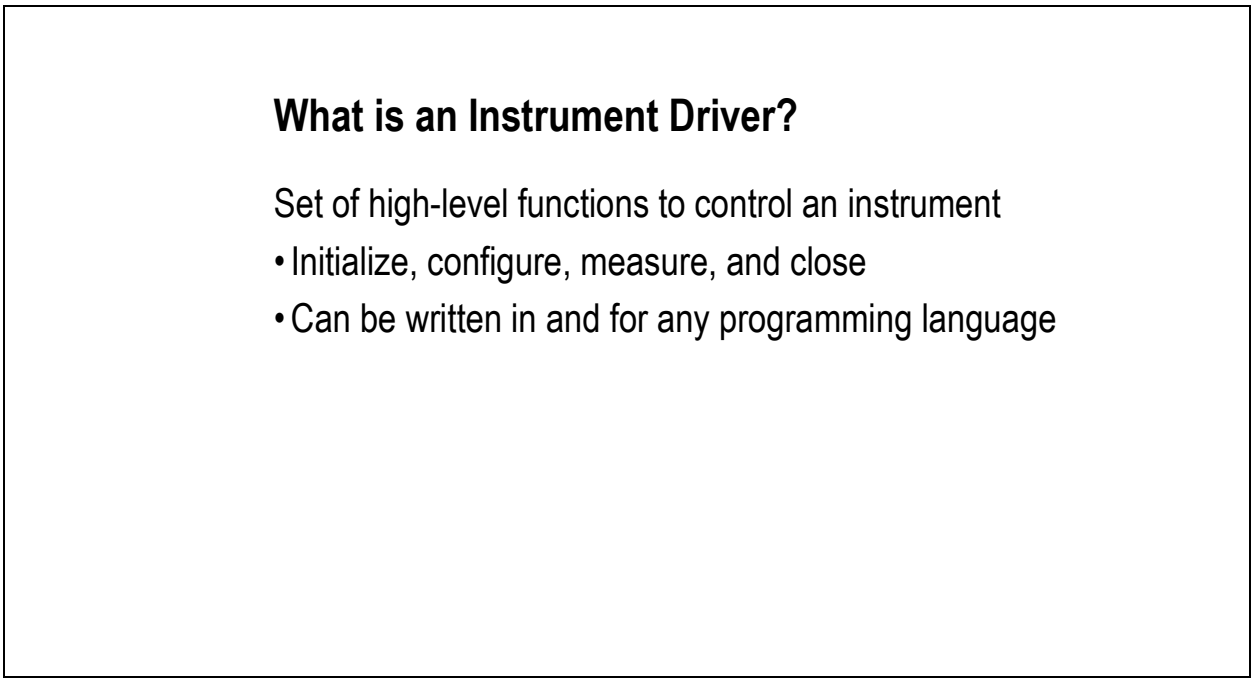
In this lesson, you will learn about:

- **Overview of Interchangeable Virtual Instruments (IVI)**
- **Advantages and Features of IVI**
- **IVI Configuration**
- **How to use IVI with TestStand**

Introduction

This section introduces and describes how you can use Interchangeable Virtual Instrument (IVI) drivers in your applications. IVI is a new instrument driver architecture with two key improvements to drivers. IVI drivers are intelligent, that is they track the state of the instrument and make run-time decisions based on that state. This improves performance. Secondly IVI drivers are interchangeable, which means you can swap instruments under the same driver.

As the tools and techniques for test system development have evolved, test developers have shifted to a more modular approach for controlling instruments. This evolution has driven test developers from writing programs with instrument commands littered throughout their code, to using modern techniques for packaging instrument I/O commands in self-contained instrument drivers. The benefits of separating instrument I/O code in instrument drivers from test logic code are tremendous—not only is it easier to write your test programs it is also easier to maintain your test code when you use instrument drivers.

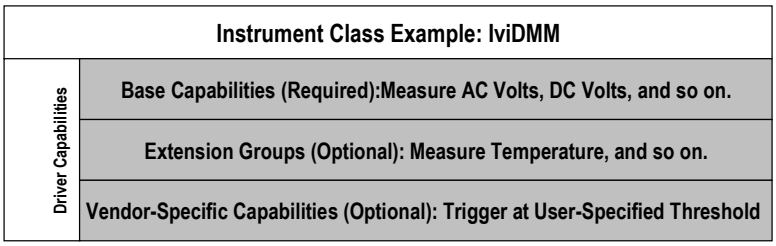


What is an Instrument Driver?

In the early days of computer controlled instrumentation systems, programmers used BASIC, C, FORTRAN, or PASCAL I/O statements in their application programs to send and receive command and data strings (or register values) to and from the various instruments to their computer through GPIB, serial, or VXI connections. Each instrument responded to a particular command that the instrument manufacturer documented in a manual, and programmers were responsible for learning these commands. Software development is often the most time consuming part of implementing an automated test system. This is especially the case when test engineers need to relearn the same command strings or register values over and over again whenever they create new applications. It quickly became apparent that programmers could save significant time and money if they wrote generic high level routines that encompassed the low level commands used to program the instrument. These reusable generic routines came to be known as instrument drivers. An important category of instrument drivers is IVI, Interchangeable Virtual Instruments, which make it possible to use the same software interface for different hardware instruments.

What is IVI?

- IVI stands for Interchangeable Virtual Instrument
- IVI Foundation was announced in August 1998
- Charter: Define standard programming interface for instrument drivers.
- IVI standard defines instrument driver class specifications:

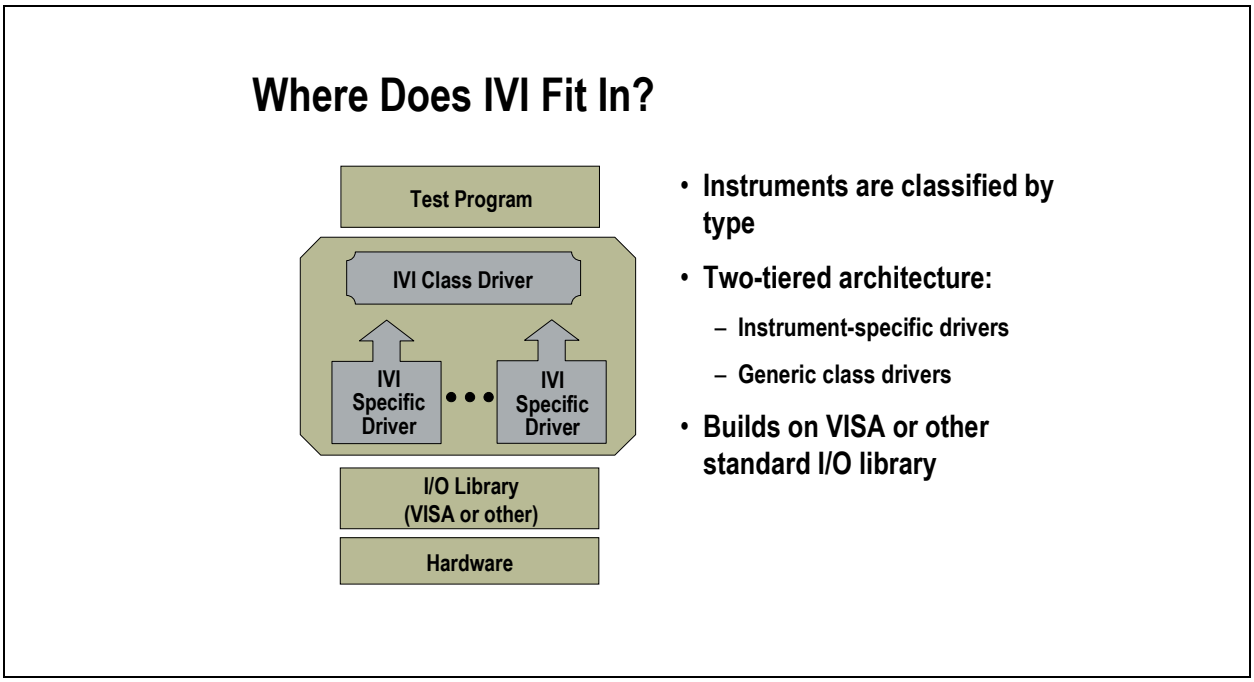


What is IVI?

In early 1997, National Instruments was approached by several instrument end-user companies interested in developing a new standard for instrument drivers that would be built on existing approaches such as *VXIplug&play* and would also address several concerns such as instrument state caching, simulation, and interchangeability.

As a result of the discussions, a group of nine companies formed the Interchangeable Virtual Instruments (IVI) Foundation. The charter of the IVI Foundation is to define a standard programming interface for instrument drivers while allowing instrument interchangeability, regardless of the instrument manufacturer or I/O bus interface.

The IVI Foundation developed the standards by identifying common instrument types such as digital multimeters or oscilloscopes. Then they identified a common set of features within each instrument type and divided the features into three groups—base capabilities, extension capabilities, and vendor specific capabilities. The IVI Foundation then developed a set of specifications for each of these instrument types, and defined that as an instrument class.



Where Does IVI Fit In?

The IVI Foundation defined a hierarchical and layered architecture for its drivers that leverage industry-standard technology. This architecture fits into a complete modular system from National Instruments. The test program uses IVI drivers for the instrument communication. The IVI layer in turn uses an industry-standard I/O library such as VISA to communicate with the hardware that is in the system, regardless of whether it is GPIB, VXI, PXI, or computer-based instrument hardware.

The IVI layer itself is divided into two sub-layers. The lower is the IVI-specific driver layer. These are instrument specific drivers that are used with specific instrument models. They are designed to provide the performance improvements of IVI, such as state-caching, multithreading, and simulation.

The upper sub-layer is the IVI class driver layer. These are the generic class drivers that you can use in conjunction with specific drivers to control any instrument in a class. This layer provides the interchangeability benefit of IVI.

The IVI specs also allow drivers to be created for instruments that do not fall into one of the IVI defined classes. You can develop specific drivers to take advantage of performance benefits or define your own instrument class.

Advantages and Features Of IVI

- Advantages
  - State caching
  - Range checking
  - Simulation
- Features
  - IVI Classes
  - IVI Attribute Model

IVI Drivers

IVI drivers have intelligent built-in features such as state caching, range checking, and simulation. IVI drivers provide attributes and classes to improve standardization and instrument interchangeability. The following section describes each of these topics in detail.

State Caching

- Physical state of instrument maintained in a software cache
- High-level functions group attributes together
  - Communication with instrument is only performed when the applicable attribute's value is valid
  - Minimizes redundant I/O to instrument
  - Minimizes costly instrument command parsing

State Caching

The attribute model of IVI drivers is a key to implement state caching. Each physical state or attribute of an instrument is maintained as an attribute in a software cache. High-level functions, group a number of attributes together and communicate with an instrument when a particular cached attribute value becomes invalid. This eliminates both redundant I/O and instrument re-configuration.

Range Checking

- Drivers verify the values specified for an attribute
  - Traditional drivers indicate valid range through online documentation
  - IVI drivers provide this information and verify the entries in an application
- Attribute dependencies are handled by driver
  - For example, Probe attenuation and vertical range

Range Checking

IVI drivers verify the values for an attribute such as a range in software, and return an error before communicating with the instrument. Traditional drivers, document valid ranges for parameters or settings, but do not provide any error checking or trapping.

IVI drivers also take into account any attribute interdependencies. For example, the valid vertical range for an oscilloscope is usually dependent upon the probe attenuation for the probe that is connected to a particular channel. IVI drivers can dynamically calculate the valid vertical range by checking the probe attenuation

Simulation

- Simulation mode does not require a physical instrument
- You can verify values to be sent to instruments while developing the test application
- Simulated data is
  - Random data with instrument specific drivers
  - User-defined data with class drivers

Simulation

Simulation eliminates the need to have an actual instrument present when you develop an application. With range checking and simulation you can verify the correctness of the test application even before purchasing the instrument or deploying a system on the test floor.

Simulation allows you to verify the values being sent to the instrument through range checking. It also simulates the data being returned from instruments. Thus, you can write the entire application, test it through simulation, and spend very little time testing with the actual hardware before deploying the system.

Instrument Classes

- Define a standard API for programming instruments
- Allow the development of hardware-independent test programs
- No source code change or recompilation to switch instruments
- IVI Foundation establishes class specifications
  - **www.ivifoundation.org**

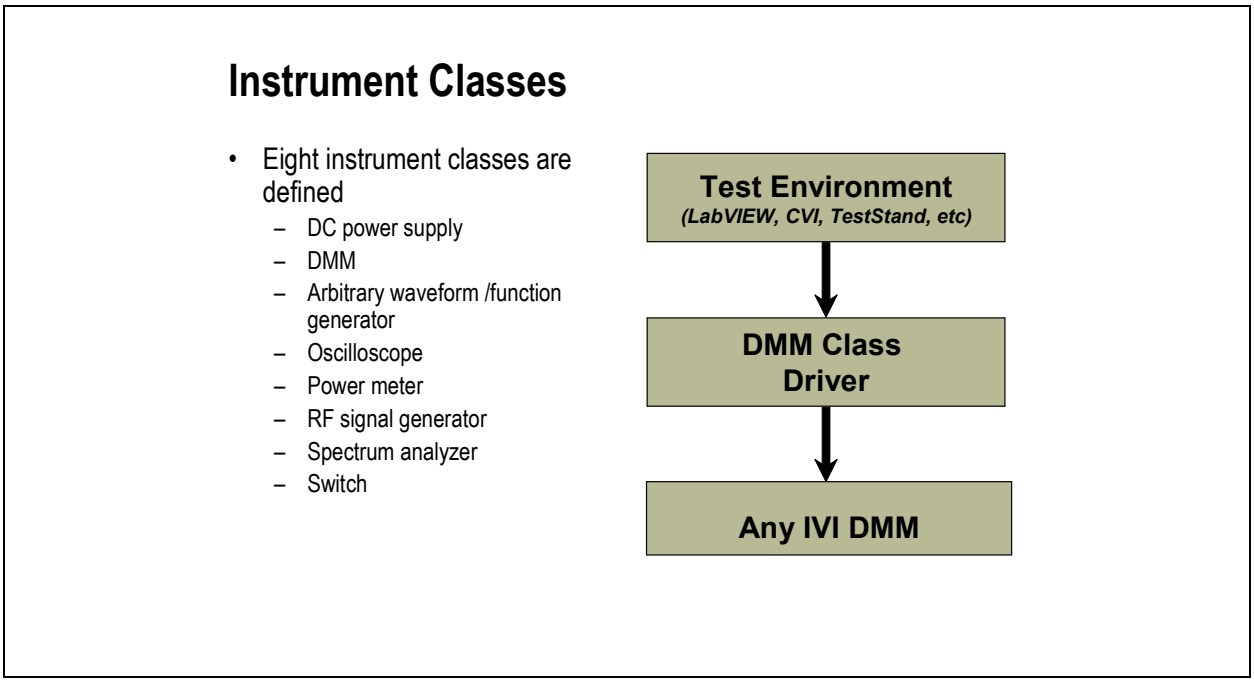
Instrument Classes

The interchangeable side of IVI drivers has been made possible by the standardization of the internal structure of instrument drivers.

The IVI class specifications divide instrument features into fundamental capabilities and extension groups.

Fundamental capabilities are attributes and functions that are commonly supported by the majority of instruments in a class. For example, range setting for a DMM or probe attenuation for a scope. This defines the basic functionality that can be interchanged among instruments in a particular class.

Extension groups are for less common features that are popular in a large number of instruments. For example, different scopes have different triggering protocols, so glitch triggers, runt triggers, and so on are considered extension groups.

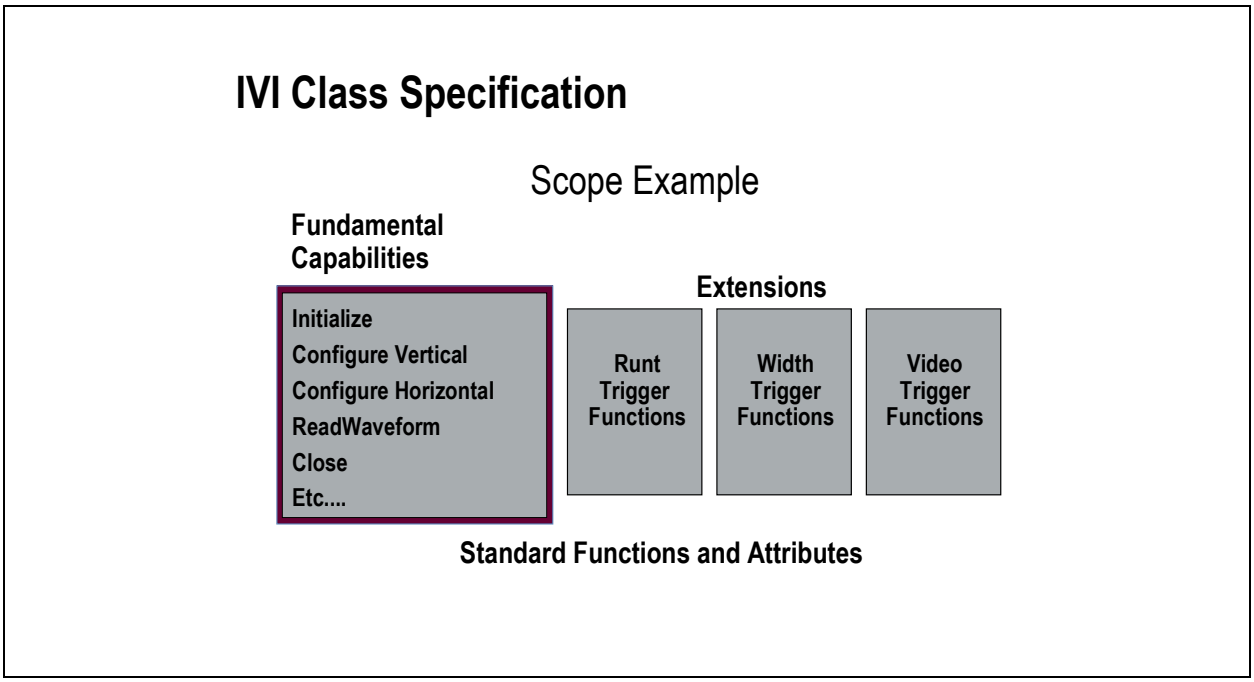


IVI Foundation Goal: Generic Interchangeable Drivers

The Interchangeable Virtual Instruments (IVI) Foundation is a group of end-user companies—System integrators and Instrument and software vendors formed at NIWeek ’98 in Austin, Texas.

The IVI Foundation defines instrument classes and a standard API for programming instruments that belong to a particular category. Using this standard, test engineers can build hardware-independent test systems. This industry-standard API has many far-reaching benefits, including:

- **Long-term test code maintenance**—Test system developers in the military and in the aerospace industries, who must maintain test systems and code for many years, can easily reuse their test code on new equipment when instruments improves or becomes obsolete.
- **Ease of programming resulting in faster code development**—All instruments in a class are exactly programmed in the same way. This procedure saves time, as developers do not need to learn a new API for every instrument they want to integrate.
- **Code reusability**—Different departments in a large corporation can share and reuse test code without being forced to use the same instrumentation hardware. In addition, manufacturers in competitive, high-volume industries, such as telecommunications or consumer electronics, can keep their production lines running when instruments malfunction or must be recalibrated.



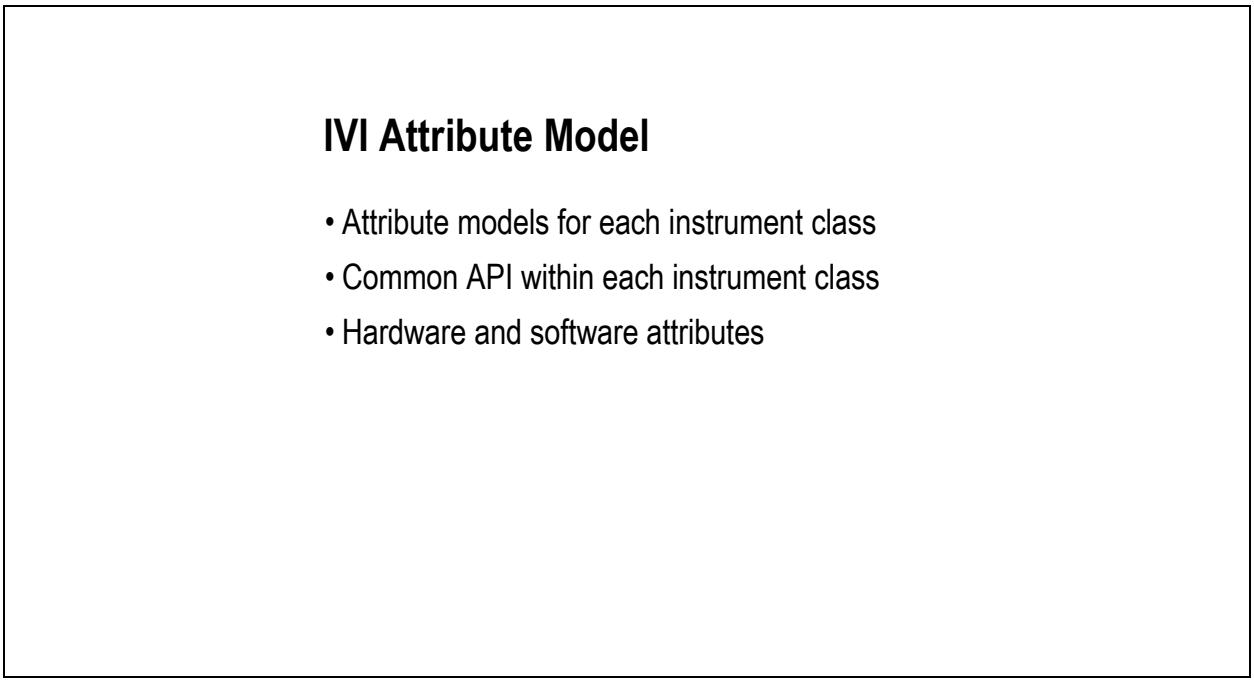
IVI Class Specification

The IVI class specifications divide instrument features into fundamental capabilities and extension groups. Attributes and functions that are commonly supported by the majority of instruments in a class, such as range setting for a DMM or probe attenuation for a scope, are considered fundamental capabilities. Fundamental capabilities define the basic functionality that can be interchanged among instruments in a particular class.

Fundamental capabilities are common functions, attributes, and attribute values supported by more than 95 percent of the instruments in a given class and must be implemented in all instrument-specific IVI drivers in the class. Extensions are less common capabilities and are optional. For example, the Arb extension is supported only by true arbitrary waveform generators.

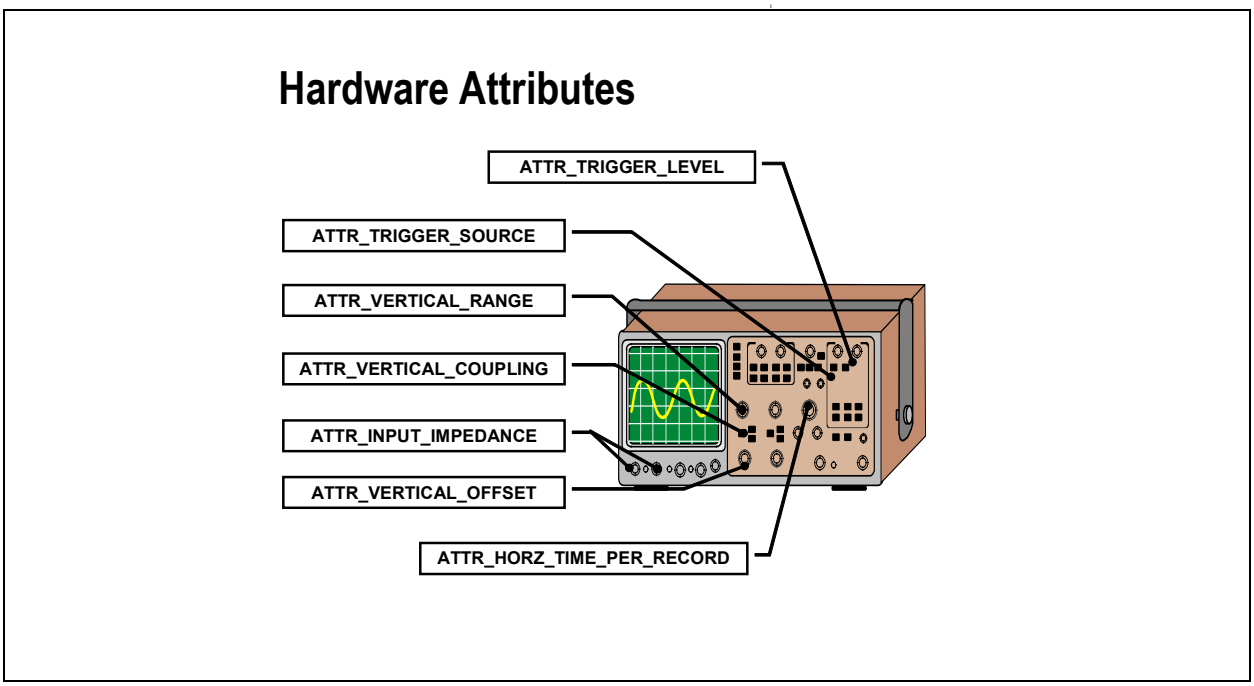
Two examples that show the difference between fundamental capabilities and extensions are described here in reference to the Scope and DMM class drivers:

- **Scope**—This class defines the vertical and horizontal subsystems as part of the fundamental capabilities. Different oscilloscopes support different triggering mechanisms and protocols, and the Scope class defines the most common as extended features.
- **DMM**—This class defines trigger and multipoint measurement functions as extended capabilities. Less common features such as autoranging capabilities and measurement functions such as light intensity are also included in the extensions.



IVI Attribute Model

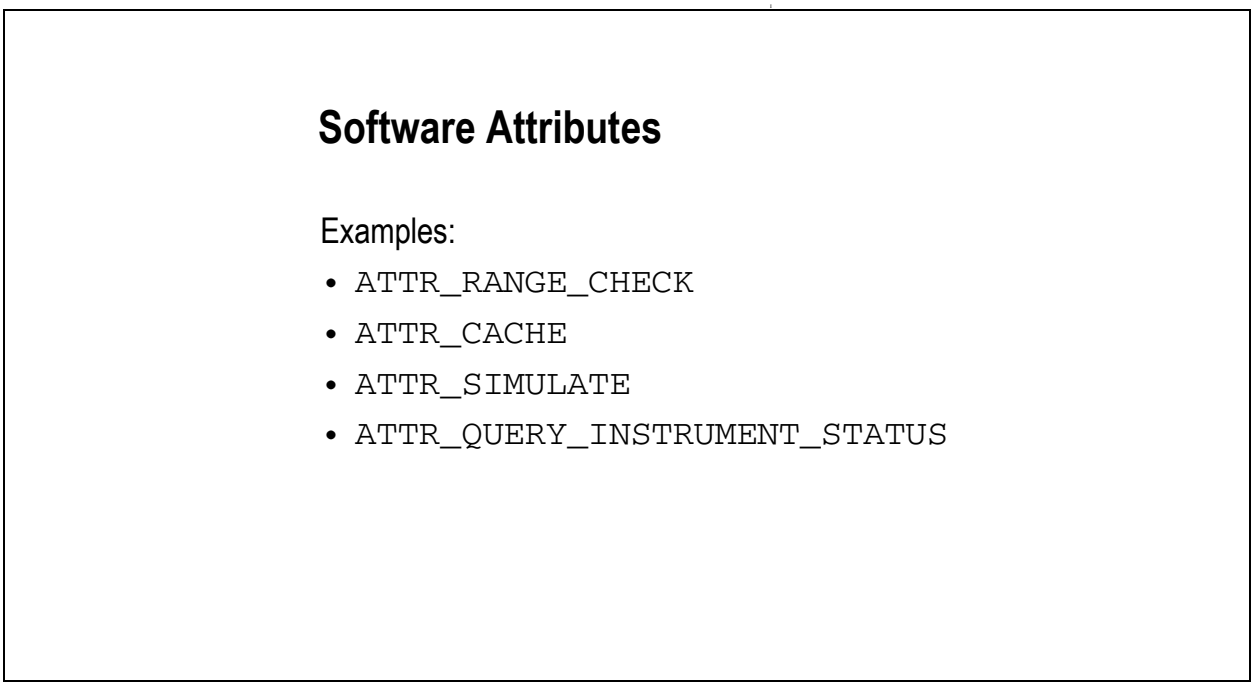
IVI defines attribute model specifications for eight categories of instruments. These specifications present a common attribute model and API for programming a particular class of instruments. Thus, this model makes the test system developer's job easier since he or she does not need to learn a different API for programming similar instruments.



Hardware Attributes

IVI drivers use a state caching model in which each instrument hardware configuration setting is associated with a unique attribute. An oscilloscope, for example, has configuration settings for vertical range, vertical offset, etc, and respective IVI attributes to cache these settings. Each attribute sets or retrieves the value of its respective hardware configuration setting.

Instruments may have several input or output acquisition channels, and each channel may have independent hardware settings. For example, a function generator may have a different DC offset voltage setting for each output channel. IVI drivers use channel-based attributes for these situations and the IVI Engine maintains cached values for each channel.



Software Attributes

IVI drivers also have associated software attributes. These attributes are not associated with any instrument features, but are useful during code development and testing.

Examples of software attributes include:

- **Range Check Attribute**—Checks values before they are sent to the instrument to see if they are valid.
- **Cache**—Turns state caching on or off.
- **Simulate**—Simulates the instrument. The instrument does not need to be present physically when simulation is turned on.
- **Query Instrument Status**—Checks the instrument error queue after every high-level driver call.

Benefits of Attribute Model

There are two main benefits:

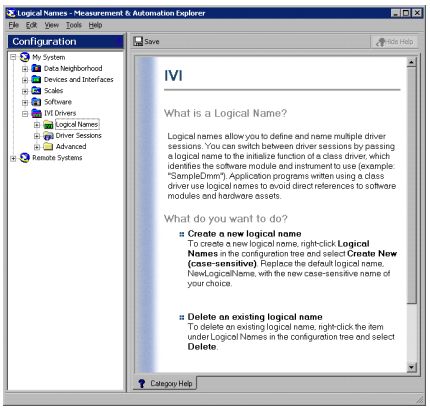
- Improved performance through state caching
- Hardware interchangeability within each class

Benefits of Attribute Model

A standard attribute model provides many benefits. You can use a state management engine to keep track of the value of each attribute, thus allowing the instrument driver to behave intelligently and avoid sending redundant configuration commands. Additionally, a standard attribute model means you can develop a software layer that allows interchanging instruments with similar capabilities without changing the test code.

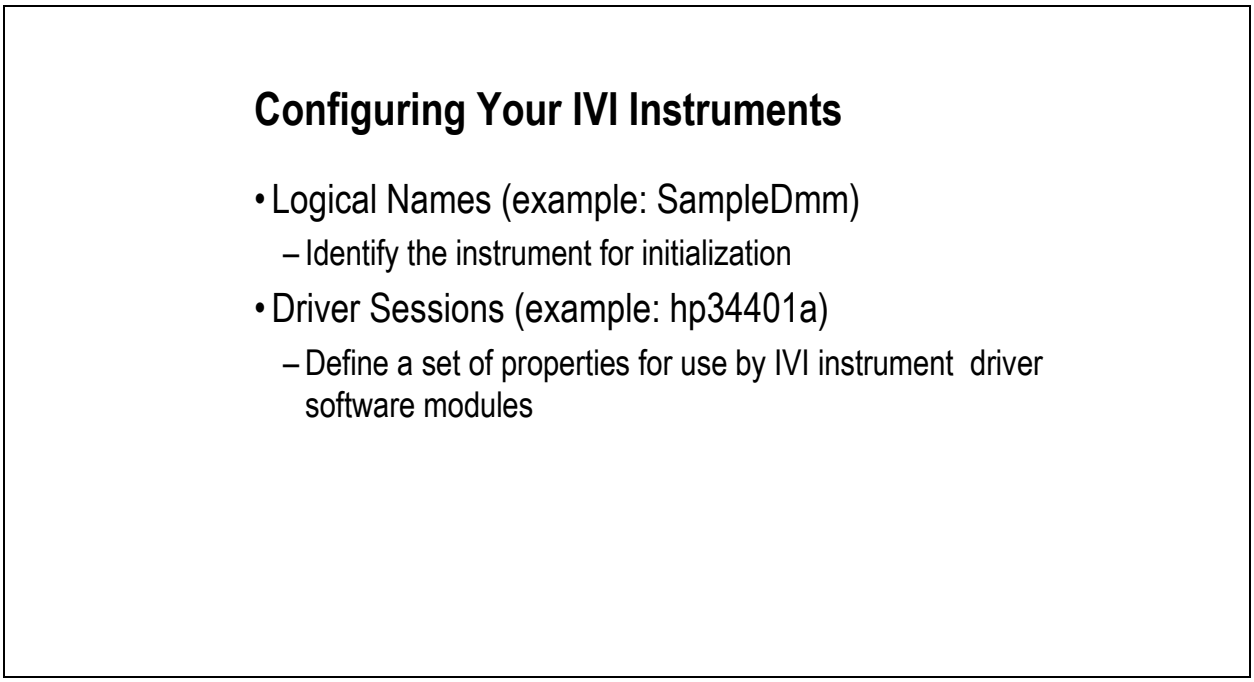
IVI Configuration

- Measurement & Automation Explorer (MAX) configures IVI systems



IVI Configuration

The IVI Driver Toolset installs MAX. MAX has several views associated with it, including one for configuring IVI instrument drivers.



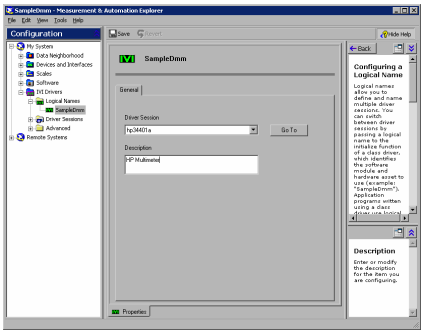
Configuring Your IVI Instruments

With MAX, you can configure the following:

- **Logical Names**—A logical name references the driver session you want to use in your application.
- **Driver Sessions**—A driver session defines a set of properties for use by IVI instrument driver software modules, such as initial configurable settings for attributes, virtual name mappings, simulation settings, and so on. By changing the driver session logical name references, you can swap instruments without changing your program.

Logical Names

- Used by class driver Initialize functions
- Associated with a specific instrument

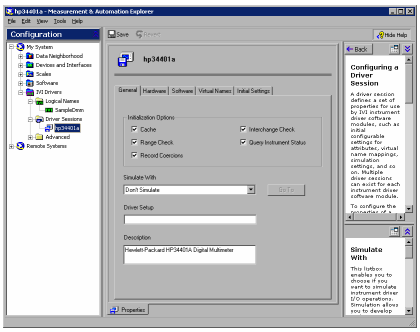


Logical Names

Logical names allow you to define and name multiple driver sessions. You can switch between driver sessions by passing a logical name to the initialize function of a class driver, which identifies the software module and instrument to use (example: `SampleDmm`). Application programs written using a class driver use logical names to avoid direct references to software modules and hardware assets.

Driver Sessions

- Define a set of properties for use by IVI instrument driver software modules

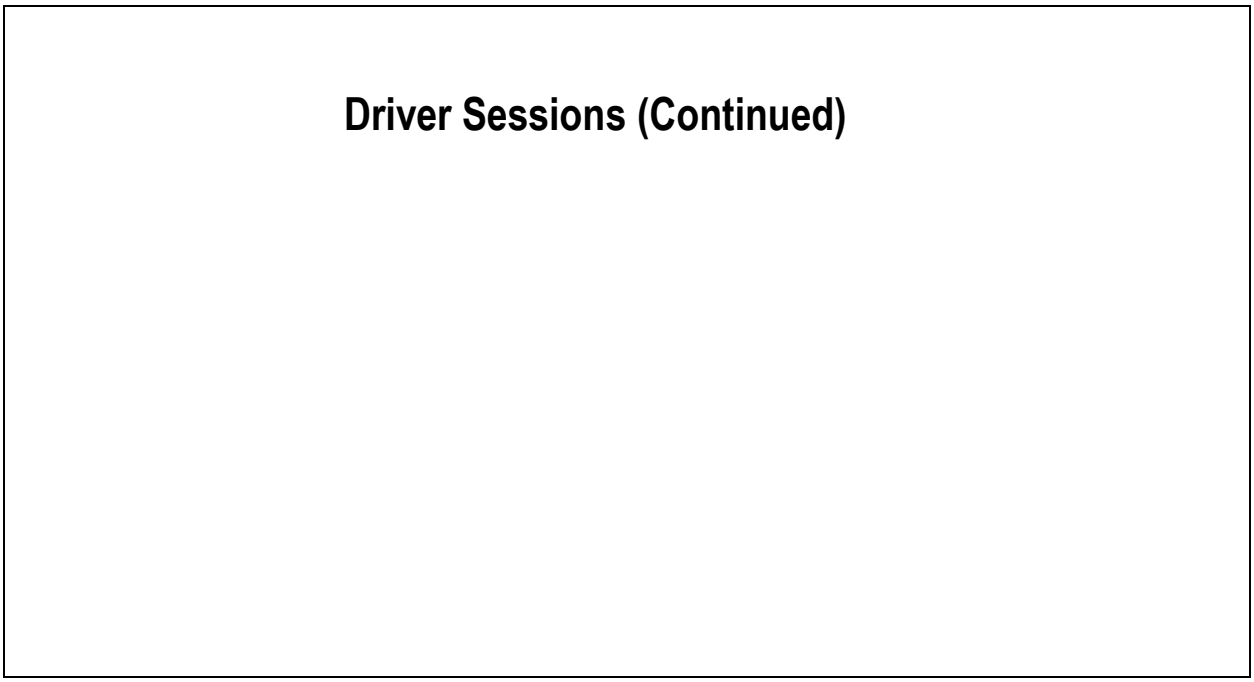


Driver Sessions

Each driver session entry contains information regarding the instrument initialization options, hardware asset, software module, virtual names, and initial settings.

Multiple driver sessions can exist for each instrument driver software module.

- **General Tab**:
  - Initialization options—This section allows you to enable or disable inherent attributes such as cache, range check, and so on.
  - Simulation—This field enables you to choose if you want to simulate instrument driver I/O operations.
  - Driver Setup—This field allows you to specify a particular instrument model to use. Drivers that support multiple instrument models must be configured with an optional parameter that identifies the specific instrument model used. Class drivers use the Driver Setup field to associate a particular instrument with a “family” driver. In a specific driver, you would specify this parameter as part of the Initialize With Options function with the option string set to DriverSetup=`Model:`*`XXX`*, where *`XXX`* is the model. For example, DriverSetup=`Model:TDS 340A` specifies that you are using the TDS 340A with the Tektronix 300 series instrument driver.



Driver Sessions (Continued)

- **Hardware Tab**:

  The Hardware Asset field allows you to choose which physical instrument you want to use with the particular driver session. If the simulation option under the General tab is enabled, select **None** for the hardware asset.

- **Software Tab**:

  The Software tab contains information about the instrument driver such as prefix of the driver, driver location, and supported instrument models.

- **Virtual Names Tab**:

  The Virtual Names tab allows you to map the physical names of your instrument with the associated virtual names. IVI uses virtual names to allow the application program to use one set of names for all possible specific drivers. For example, an IVI specific driver might name channels A, B, and C, while you might choose to name them Chan1, Chan2, and Chan3. In this example, A, B, and C are physical names defined by the software module. Chan1, Chan2, and Chan3 are the corresponding virtual names used by the application program.

- **Initial Settings Tab**:

  The Initial Settings tab allows you to select instrument's attributes for which you can set the initial values.

IVI Configuration Demo

Open the IVI Drivers section of MAX

1. Create a Driver Session
2. Assign the Driver Session a software module and a simulation driver
3. Create a Logical Name
4. Assign the Driver Session to the Logical Name
5. Save the Configuration

IVI Configuration Demo

Note The demos in this lesson require the IVI Compliance package, and an IVI compliant DMM driver (such as NI-DMM). The IVI Driver Toolset is suggested but not required. No hardware is needed.

Complete the following steps to configure the IVI driver.

1. Open **MAX**.
2. Open **IVI Drivers**.
3. Right-click **Driver Sessions** and select **Create New**.
4. Name the Driver Session `DMMDriver`.
5. On the **General** tab, select **nisDMM** from the **Simulate With** drop-down menu.

   nisDMM is a simulation driver which is installed with the IVI Driver toolset. This driver is useful because it allows you to configure the simulated data that will be returned. If you don't have nisDMM as an option then you do not have the IVI Driver Toolset installed. You should be able to use Specific Driver instead, but you will not be able to select the simulation values.
6. On the **Software** tab, select **NIDMM** from the **Software Module** drop-down menu.

   You should be able to use another IVI compliant DMM driver in place of NIDMM. If you have more than one IVI compliant DMM Driver

installed (for example, the hp34401a) then you can complete the demo with one driver and then switch to the other and show that the TestStand program still works without any changes.

IVI Configuration Demo (Continued)

IVI Configuration Demo (Continued)

7. Right-click **Logical Names** in the **Configuration** tree and select **Create New**.
8. Name the Logical Name `IVIDMM`.
9. Select **DMMDriver** from the **Driver Session** drop-down menu.
10. Click **Save IVI Configuration**.

Using IVI With TestStand

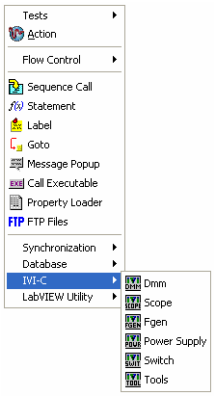
- Option 1: IVI Step Types
  - Reduces programming
  - No external code module needed
- Option 2: IVI Through Code Modules
  - Easier to analyze or manipulate data
  - Easier to synchronize or communicate between instruments
  - Allows the use of instrument specific drivers and functions
  - Allows access to more IVI classes

Using IVI With TestStand

When using IVI instruments in a TestStand system, you can control the instruments in one of the two ways. TestStand provides a set of IVI Step Types which allow you to directly control the IVI instrument. When controlling an instrument with these steps, you save programming time because you do not need to create a separate code module. The IVI step types provide access to all of the basic functions for most classes of instrument driver. However, the IVI step types do not provide access to all instrument classes, and cannot be used with instrument specific drivers or functions. The second option for controlling an IVI compliant instrument is to write a code module to interface to the instrument. For example, you could write a LabVIEW VI to interface with an IVI driver and then call the LabVIEW VI as a code module from TestStand. This option requires you to write more code, but is more flexible, as a code module can access any IVI driver or function. When you need to communicate between instruments or analyze and manipulate data it is often easier to write a code module as this simplifies data passing and synchronization.

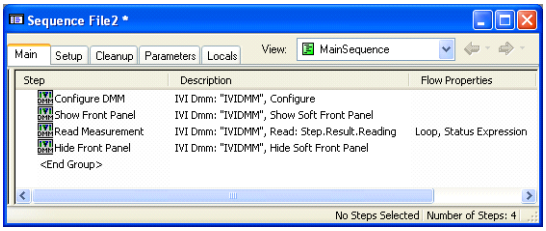
IVI Step Types

- TestStand provides steps for directly accessing IVI instruments
- IVI Step types use the IVI Class drivers
- Each step type contains a number of functions



IVI Step Types Demo

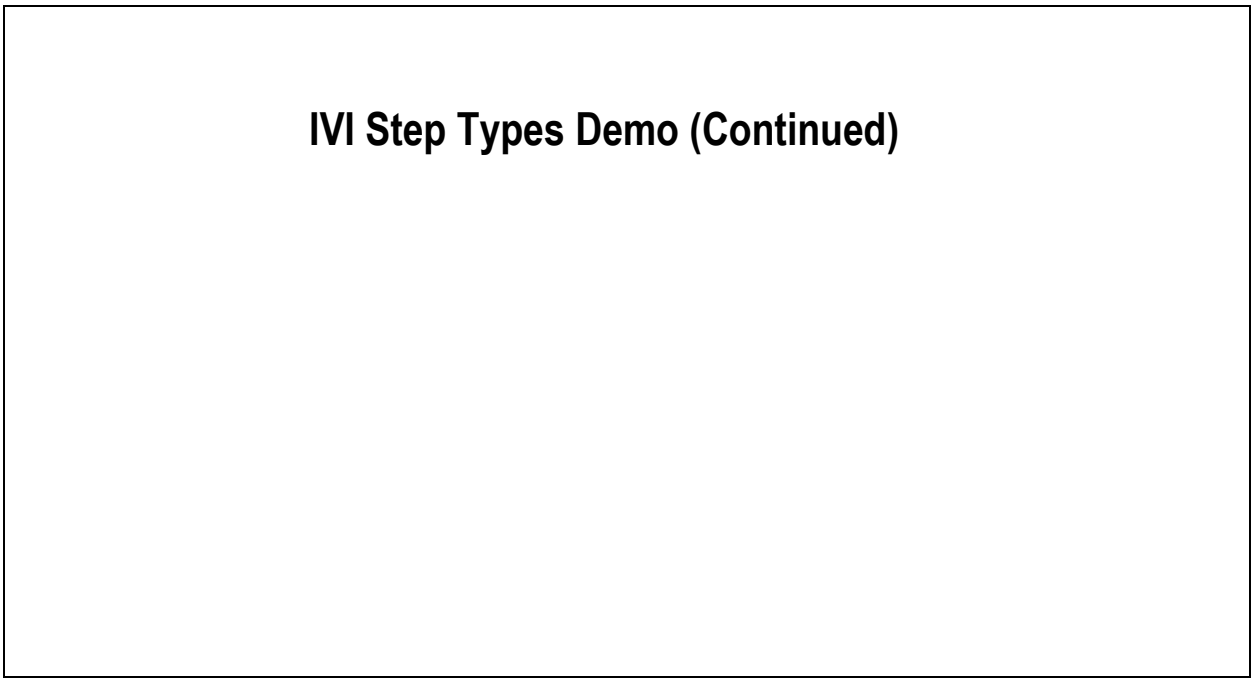
1. Configure the DMM
2. Show the Soft Front Panel
3. Read and test a measurement in a loop
4. Hide the Soft Front Panel
5. Run the sequence



IVI Step Types Demo

Note You must configure an IVI driver session or logical name before doing this demo.

1. Configure **DMM**.
   a. Insert a **Dmm** step from the IVI group.
   b. Name the step `Configure DMM`.
   c. Right-click the **Configure DMM** step and select **Edit IVI Dmm**.
   d. Set the **Logical Name** to **IVIDMM** using the drop-down menu.
      IVIDMM is the logical name created in the previous demo.
   e. Leave the default values (Operation: Configure) and click **OK** twice.
2. Show the Soft Front Panel.
   a. Copy the **Configure DMM** step and name the copy `Show Front Panel`.
   b. Right-click the **Show Front Panel** step and select **Edit IVI Dmm**.
   c. Select **Show Soft Front Panel** from the **Operation** drop-down menu.
   d. Leave the default values and click **OK** twice.



IVI Step Types Demo (Continued)

3. Read and Test Measurement.
   a. Copy the **Configure DMM** step and name the copy `Read Measurement`.
   b. Right-click the **Read Measurement** step and select **Edit IVI Dmm**.
   c. Select **Read** from the **Operation** drop-down menu.
   d. Leave the default values and click **OK**.
   e. Click the **Loops** tab and configure the following options:

      Loop Type: Fixed number of loops

      Number of Loops: 50

      Loop result is Fail if < 50 **% of iterations Pass**
   f. Click the **Expressions** tab and enter the following **Status Expression**:

      ```
      (Step.Result.Reading.Data < 10)?"Passed":"Failed"
      ```
   g. Click **OK**.
4. Close the Soft Front Panel.
   a. Copy the **Configure DMM** step and name the copy `Hide Front Panel`.
   b. Right-click the **Hide Front Panel** step and select **Edit IVI Dmm**.

c. Select **Hide Soft Front Panel** from the **Operation** drop-down menu.

d. Click **OK** twice.

5. Run the sequence.

   a. Run the sequence with **Test UUTs**.

   b. Enter a serial number.

   c. When the **IviDmm Simulator Setup** dialog box appears, leave the default values and click **OK**.

      The IviDmm Simulator comes from the nisdmm simulator that you configured in the configuration demo. This simulator allows you to define the nature of the simulated data.

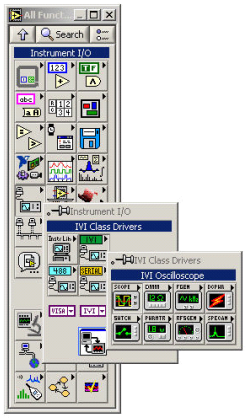
   d. Run the sequence a couple of times and notice that the status is determined by the value of the random readings.

IVI Code Modules

- IVI Drivers provide DLLs which can be called from most programming environments including CVI, LabVIEW, Visual Basic and Visual C++
- Most IVI Drivers also provide specific CVI Function Panels and/or LabVIEW VIs
- The IVI Class Drivers provide full support from multiple languages

IVI Code Modules: LabVIEW

- LabVIEW IVI Class Drivers are installed with the IVI Compliance packages
- Most instrument specific IVI drivers provide LabVIEW VIs
- Others can be used in LabVIEW by converting a CVI function panel or by calling DLLs

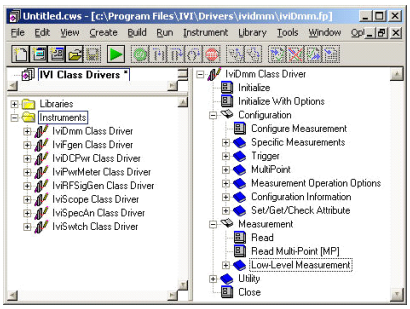


IVI Code Modules: LabVIEW

The example at `C:\Program Files\National Instruments\LabVIEW 7.1\examples\instr\IviClass.llb\IviDmm - Single Point Measurement.vi` is a good example which you can run using the configuration from the previous demos.

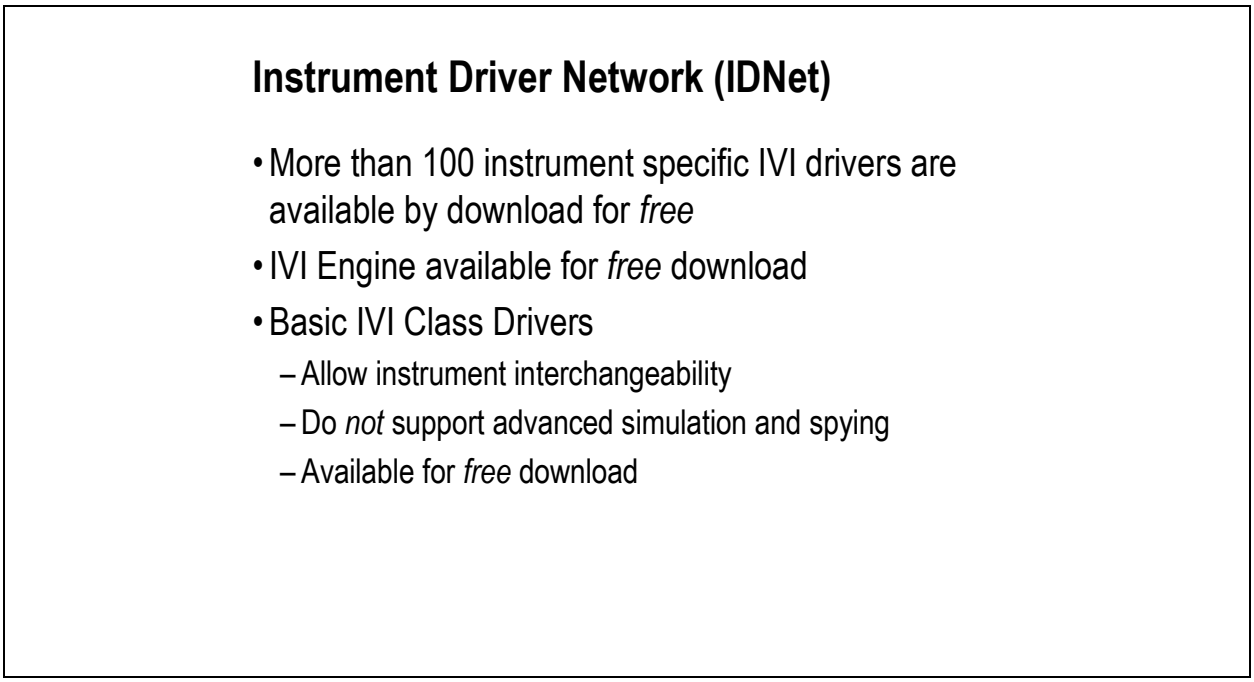
IVI Code Modules: LabWindows/CVI

- IVI Class drivers provide function panels in *Program Files\IVI\Drivers*
- Most instrument specific IVI drivers provide CVI function panels
- Other drivers can be accessed with DLL calls



IVI Code Modules: LabWindows/CVI

The example at `C:\Program Files\IVI\Drivers\ividmm\Examples\CVI\ividmm.prj` is a good example which you can run using the configuration from the previous demos.



Instrument Driver Network (IDNet)

At `ni.com/idnet`, you can obtain information about IVI to:

- Download drivers.
- Obtain the IVI engine.
- Get information on instrument driver standards.
- Obtain basic IVI class drivers.

At `ni.com/ivi` you can obtain information about IVI to:

- Get information on IVI drivers.

At `www.ivifoundation.org` you can obtain information about IVI to:

- Get information on IVI class specifications.

Notes

Notes

B

Additional Information and Resources

This appendix contains additional information about National Instruments technical support options and TestStand resources.

National Instruments Technical Support Options

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services.

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

    For information about other technical support options in your area, visit `ni.com/services` or contact your local office at `ni.com/contact`.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. The NI Alliance Partners joins system integrators, consultants, and hardware vendors to provide comprehensive service and expertise to customers. The program ensures qualified, specialized assistance for application and system development. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Other National Instruments Training Courses

National Instruments offers several training courses for TestStand users. These courses continue the training you received here and expand it to other areas. Visit `ni.com/training` to purchase course materials or sign up for instructor-led, hands-on courses at locations around the world.

National Instruments Certification

Earning an NI certification acknowledges your expertise in working with NI products and technologies. The measurement and automation industry, your employer, clients, and peers recognize your NI certification credential as a symbol of the skills and knowledge you have gained through experience. areas. Visit `ni.com/training` for more information about the NI certification program.

Course Evaluation

Course _____

Location _____

Instructor _____ Date _____

Student Information (optional)

Name _____

Company _____ Phone _____

Instructor

Please evaluate the instructor by checking the appropriate circle.	Unsatisfactory	Poor	Satisfactory	Good	Excellent
Instructor's ability to communicate course concepts	❍	❍	❍	❍	❍
Instructor's knowledge of the subject matter	❍	❍	❍	❍	❍
Instructor's presentation skills	❍	❍	❍	❍	❍
Instructor's sensitivity to class needs	❍	❍	❍	❍	❍
Instructor's preparation for the class	❍	❍	❍	❍	❍

Course

Training facility quality	❍	❍	❍	❍	❍
Training equipment quality	❍	❍	❍	❍	❍

Was the hardware set up correctly? ❍ Yes ❍ No

The course length was ❍ Too long ❍ Just right ❍ Too short

The detail of topics covered in the course was ❍ Too much ❍ Just right ❍ Not enough

The course material was clear and easy to follow. ❍ Yes ❍ No ❍ Sometimes

Did the course cover material as advertised? ❍ Yes ❍ No

I had the skills or knowledge I needed to attend this course. ❍ Yes ❍ No If no, how could you have been better prepared for the course? _____

_____

What were the strong points of the course? _____

_____

What topics would you add to the course? _____

_____

What part(s) of the course need to be condensed or removed? _____

_____

What needs to be added to the course to make it better? _____

_____

How did you benefit from taking this course? _____

_____

Are there others at your company who have training needs? Please list. _____

_____

_____

Do you have other training needs that we could assist you with? _____

_____

How did you hear about this course? ❍ NI Web site ❍ NI Sales Representative ❍ Mailing ❍ Co-worker ❍ Other _____