



NATIONAL INSTRUMENTS™
TestStand™

Getting Started with TestStand

Internet Support

E-mail: support@natinst.com

FTP Site: <ftp.natinst.com>

Web Address: <http://www.natinst.com>

Bulletin Board Support

BBS United States: 512 794 5422

BBS United Kingdom: 01635 551422

BBS France: 01 48 65 15 59

Fax-on-Demand Support

512 418 1111

Telephone Support (USA)

Tel: 512 795 8248

Fax: 512 794 5678

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

Important Information

Warranty

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

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

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

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

Copyright

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

Trademarks

CVI™, LabVIEW™, natinst.com™, National Instruments™, the National Instruments logo, and TestStand™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Organization of This Manual	xi
Conventions Used in This Manual	xii
Customer Communication	xiii

Chapter 1

Introduction to TestStand

Installing TestStand	1-1
Minimum System Requirements	1-1
Installing TestStand	1-1
What the Setup Programs Install	1-2
Learning TestStand	1-3
TestStand System Overview	1-4
Major Software Components of TestStand	1-4
TestStand Sequence Editor	1-6
TestStand Run-Time Operator Interfaces	1-6
TestStand Test Executive Engine	1-7
Module Adapters	1-7

Chapter 2

Loading and Running Sequences

Starting TestStand	2-1
Introduction to the Sequence Editor	2-2
Menu Bar	2-2
Toolbar	2-3
Development Workspace	2-3
Status Bar	2-3
Loading a Sequence File	2-4
About Sequences	2-6
Running a Sequence	2-7
Setting Up Tracing Options	2-7
Running a Sequence Directly	2-8
Running a Sequence Using the Process Model	2-11

Chapter 3

Editing Steps in a Sequence

Setting Up the Example.....	3-1
Adding a New Step.....	3-1
Specifying the Test Module.....	3-4
Changing Step Properties	3-5
Adding a SubSequence to a Sequence.....	3-13

Chapter 4

Debugging Sequences

Setting Up the Example.....	4-1
Step Mode Execution	4-1

Chapter 5

Using Variables and Properties

Setting Up the Example.....	5-1
Using TestStand Variables	5-2
Using the Context Tab.....	5-8
Using the Watch Expression Pane.....	5-10

Chapter 6

Creating and Debugging Tests

Debugging a LabVIEW Standard Prototype Virtual Instrument	6-1
Setting Up the Example	6-1
Creating a Virtual Instrument Code Module.....	6-2
Debugging a Virtual Instrument Code Module.....	6-10
Debugging a LabWindows/CVI Standard Prototype Module.....	6-11
Setting Up the Example	6-11
Creating a C/CVI Code Module Test.....	6-12
Debugging a CVI Code Module	6-21

Chapter 7

Using Run-Time Operator Interfaces

Loading Sequences.....	7-1
Running and Debugging Sequences.....	7-4
Running Multiple Executions.....	7-6

Chapter 8

Using Callbacks

Setting Up the Example	8-1
Overriding a Process Model Callback	8-1

Chapter 9

Adding Users and Setting Privileges

Setting Up the Example	9-1
Using the User Manager	9-1

Chapter 10

Using ActiveX in Code Modules

Using ActiveX in LabVIEW Test Virtual Instruments	10-2
Setting Up the Example	10-2
Creating the Sequence and Virtual Instrument Tests	10-2
Running the Sequence	10-12
Using ActiveX in LabWindows/CVI Code Modules	10-14
Setting Up the Example	10-14
Creating the Sequence and Tests	10-14
Running the Sequence	10-21

Chapter 11

Additional Development Features

Setting Up the Example	11-1
Interactive Execution	11-1
Running Selected Steps as a Separate Execution	11-1
Running Selected Steps During an Execution	11-3
Calling Sequences Dynamically and Passing Parameters	11-5
Adding a Step to Sequence	11-5
Running a Sequence	11-9

Chapter 12

Customizing the Report

Setting Up the Example	12-1
Configuring Test Report Options	12-1
Using External Report Viewers	12-5
Adding to a Report	12-6

Chapter 13

Converting LabVIEW and LabWindows/CVI Test Executive Sequences

Converting LabVIEW Test Executive Sequences.....	13-1
Converting LabWindows/CVI Test Executive Sequences.....	13-2

Appendix A

Customer Communication

Glossary

Figures

Figure 1-1.	TestStand System Architecture.....	1-5
Figure 2-1.	Login Dialog Box	2-1
Figure 2-2.	Sequence Editor Main Window	2-2
Figure 2-3.	Sequence Editor Toolbar.....	2-3
Figure 2-4.	Sequence Editor Status Bar.....	2-3
Figure 2-5.	Open Dialog Box	2-4
Figure 2-6.	Sample1.seq Sequence File Window	2-5
Figure 2-7.	Sequence File View Ring.....	2-6
Figure 2-8.	Execution Tab on the Station Options Dialog Box.....	2-8
Figure 2-9.	Sample1.seq Execution Window	2-9
Figure 2-10.	Test Simulator Dialog Box	2-10
Figure 2-11.	Report Generating Status Bar	2-12
Figure 3-1.	Selecting the Module Adapter	3-2
Figure 3-2.	Inserting a New Step	3-3
Figure 3-3.	Specify Module Dialog Box	3-5
Figure 3-4.	Step Properties Dialog Box.....	3-6
Figure 3-5.	Preconditions Dialog Box	3-7
Figure 3-6.	Run Options Tab	3-8
Figure 3-7.	Post Actions Tab	3-10
Figure 3-8.	Loop Options Tab	3-11
Figure 3-9.	Expressions Tab	3-12
Figure 3-10.	Edit Sequence Call Dialog Box	3-14
Figure 4-1.	Configure Message Box Step Dialog Box	4-2
Figure 4-2.	Paused Execution of Sample3.seq	4-3
Figure 4-3.	Single-Stepping Toolbar Buttons.....	4-4
Figure 4-4.	Steps View while Suspended in Subsequence	4-5

Figure 5-1.	Insert Local Context Menu Command	5-2
Figure 5-2.	Edit Statement Step Dialog Box.....	5-3
Figure 5-3.	Expression Browser Dialog Box	5-4
Figure 5-4.	Preconditions for the Loop End Step	5-7
Figure 5-5.	Context Tab	5-8
Figure 5-6.	Updated Watch Window Pane	5-11
Figure 6-1.	Adapter Configuration.....	6-2
Figure 6-2.	LabVIEW Adapter Configuration	6-3
Figure 6-3.	LabVIEW Step Module Information.....	6-4
Figure 6-4.	New Clock Frequency VI in LabVIEW	6-5
Figure 6-5.	Completed Clock Frequency.vi Front Panel	6-7
Figure 6-6.	Clock Frequency.vi Block Diagram	6-8
Figure 6-7.	Edit Numeric Limits Test Dialog Box	6-9
Figure 6-8.	LabVIEW Highlight Execution Mode	6-10
Figure 6-9.	Adapter Configuration.....	6-12
Figure 6-10.	C/CVI Standard Adapter Configuration.....	6-13
Figure 6-11.	Edit CVI Module Call—Module Tab.....	6-14
Figure 6-12.	Edit CVI Module Call—Source Code Tab.....	6-15
Figure 6-13.	Generated Result from Create Code Command	6-16
Figure 6-14.	Edit Numeric Limits Test Dialog Box	6-20
Figure 6-15.	Stepping into the GetFrequency Function.....	6-21
Figure 7-1.	LabWindows/CVI Operator Interface	7-2
Figure 7-2.	Open Sequence in Operator Interface	7-3
Figure 7-3.	Paused Execution in Operator Interface	7-5
Figure 8-1.	TestStand Model Callbacks.....	8-3
Figure 8-2.	Test UUTs Sequence	8-4
Figure 8-3.	Adding Callbacks to a Sequence	8-5
Figure 9-1.	User Manager Window	9-2
Figure 9-2.	New User Dialog Box	9-3
Figure 9-3.	Configure Privileges in New Profile	9-4
Figure 10-1.	Insert Locals Array of Numeric.....	10-3
Figure 10-2.	Array Bounds Dialog Box.....	10-3
Figure 10-3.	GenerateWaveform.vi Front Panel.....	10-5
Figure 10-4.	TestStand Control Palette.....	10-6
Figure 10-5.	GenerateTestStandWaveform.vi Control Panel	10-7
Figure 10-6.	GenerateWaveform.vi Block Diagram.....	10-8
Figure 10-7.	TestStand Function Palette	10-9
Figure 10-8.	GenerateTestStandWaveform.vi Block Diagram.....	10-9

Figure 10-9.	DisplayTestStandWaveform.vi Front Panel	10-11
Figure 10-10.	DisplayTestStandWaveform.vi Block Diagram	10-12
Figure 10-11.	Locals.Arraydata	10-13
Figure 10-12.	Insert Locals Array of Numeric	10-15
Figure 10-13.	Array Bounds Dialog Box	10-15
Figure 10-14.	Generated GenerateTestStandWaveform Source.....	10-17
Figure 10-15.	Locals.Arraydata Values	10-22
Figure 11-1.	Selecting Multiple Steps in a Sequence File Window	11-2
Figure 11-2.	Breakpoint During Interactive Execution	11-3
Figure 11-3.	Loop on Selected Steps During Execution	11-4
Figure 11-4.	Dynamically Calling with an Expression	11-8
Figure 11-5.	Dynamically Calling a Sequence	11-9
Figure 11-6.	INTELProcessor.seq in the Call Stack Pane	11-10
Figure 11-7.	Sequence Parameters in the Context Tab.....	11-11
Figure 12-1.	UUT Report Setting	12-2
Figure 12-2.	Test Report in Text Format.....	12-3
Figure 12-3.	Test Report in HTML Format.....	12-4
Figure 12-4.	Callbacks Dialog Box	12-6
Figure 12-5.	ModifyReportHeader in Sequence View	12-6
Figure 12-6.	Insert String Local in ModifyReportHeader	12-7
Figure 12-7.	Completed ModifyReportHeader Sequence	12-8
Figure 12-8.	New HTML Header	12-9

Tables

Table 1-1.	TestStand Subdirectories	1-2
Table 5-1.	First-Level Properties of the Sequence Context	5-9

About This Manual

Organization of This Manual

Getting Started with TestStand is organized as follows:

- Chapter 1, *Introduction to TestStand*, contains instructions for installing TestStand, and provides an overview of the TestStand product.
- Chapter 2, *Loading and Running Sequences*—In this chapter, you learn to load and run sequences in the TestStand sequence editor, and you learn about some of the windows in the sequence editor.
- Chapter 3, *Editing Steps in a Sequence*—In this chapter, you add a step to a sequence and then configure the properties of the step. Also, you add a subsequence call to another sequence.
- Chapter 4, *Debugging Sequences*—In this chapter, you use some of the sequence debugging features of TestStand to single-step through your sequence during an execution.
- Chapter 5, *Using Variables and Properties*, teaches you how you can use variables and properties in TestStand, and points out features of TestStand that help you monitor the values of variables and properties.
- Chapter 6, *Creating and Debugging Tests*—In this chapter, you learn how to create and debug code modules written in LabVIEW and LabWindows/CVI for TestStand.
- Chapter 7, *Using Run-Time Operator Interfaces*—In this chapter, you learn how to use the LabWindows/CVI operator interface. The features that this chapter discusses also apply to the LabVIEW and the Visual Basic operator interfaces.
- Chapter 8, *Using Callbacks*—In this chapter, you learn how to customize the execution of a sequence within TestStand using callbacks.
- Chapter 9, *Adding Users and Setting Privileges*, discusses how to use the TestStand User Manager, and how you can add new users and change their privileges.
- Chapter 10, *Using ActiveX in Code Modules*, teaches you how to use ActiveX from within a code module in TestStand.
- Chapter 11, *Additional Development Features*—In this chapter, you learn how to use some of the more advanced features available when developing and debugging sequences. Also, you learn how to

interactively execute steps, and how to dynamically call a sequence by name.

- Chapter 12, *Customizing the Report*—In this chapter, you learn how to customize report generation within TestStand.
- Chapter 13, *Converting LabVIEW and LabWindows/CVI Test Executive Sequences*, explains how to convert existing test sequences from Test Executive to TestStand. TestStand provides a conversion utility for converting LabVIEW Test Executive and LabWindows/CVI Test Executive sequence files to TestStand sequence files.
- Appendix A, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

Conventions Used in This Manual

This manual uses the following conventions:

<>	Angle brackets enclose the name of a key on the keyboard—for example, <shift>.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence File»Page Setup»Options»Substitute Fonts directs you to pull down the File menu, select the Page Setup item, select Options , and finally select the Substitute Fonts options from the last dialog box.
bold	Bold text denotes the names of menus, menu items, or dialog box buttons.
<i>bold italic</i>	Bold italic text denotes a note.
<i>italic</i>	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value.
monospace	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, classes, entry points,

properties, user profiles, login names, and for statements and comments taken from programs.

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 have changed.

monospace italic

Italic text in this font denotes that you must enter the appropriate words or values in the place of these items.

paths

Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files.

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix A, *Customer Communication*, at the end of this manual.

Introduction to TestStand

This chapter contains instructions for installing TestStand, and provides an overview of the TestStand product.

TestStand is a flexible, powerful test executive framework for building, customizing and deploying a full-featured test executive system.

Installing TestStand

Before starting on your test applications, you must install TestStand on your computer. The TestStand setup program installs the software in approximately five minutes.

Minimum System Requirements

To run TestStand for Windows, you must have the following:

- Windows 95, Windows 98, or Windows NT 4.0 or later
- Personal computer using at least a 133 MHz Pentium class or higher microprocessor
- SVGA resolution (or higher) video adapter, minimally 800 × 600 video resolution
- Minimum of 64 MB of memory
- 60 MB of free hard disk space
- Microsoft-compatible mouse

Installing TestStand

Follow these instructions to install TestStand:

1. Make sure that your computer and monitor are powered on and that you have installed Windows 95, Windows 98, or Windows NT 4.0 or later.
2. Close all open Windows applications, and leave the operating system in Windows.
3. Insert the installation CD into the CD-ROM Drive.

4. Choose the **Run** option from the Desktop Taskbar.
5. Type `x:\tssetup.exe` (where *x* is the drive you are using) in the command line box and click on **OK**.
6. Follow the instructions that appear in the dialog boxes.

National Instruments recommends you install the complete TestStand program to take full advantage of all the TestStand capabilities. If you choose to install with options, select the options you want and follow the directions on the screen. You can run the setup program again, later and install additional files, if necessary.

What the Setup Programs Install

The setup program installs the TestStand development environment and a number of additional files on your system. The full installation includes example files that illustrate many of the features in TestStand and tutorial programs that you use throughout this manual. The installer installs TestStand and the associated files in subdirectories on your hard disk as shown in Table 1-1.

Table 1-1. TestStand Subdirectories

Directory Name	Contents
AdapterSupport	Support files for the LabVIEW and C/CVI Standard Prototype Adapters
Api	TestStand ActiveX Automation Server libraries for LabWindows/CVI and MFC
Bin	TestStand sequence editor executable, engine DLLs, and support files
Cfg	Configuration files for TestStand engine and sequence editor options
CodeTemplates	Source code templates for step types—This directory contains an <code>NI</code> and a <code>User</code> subdirectory
Components	Components that come with TestStand and components that you develop—This includes callback files, converters, icons, language files, process model files, step types, source files, and utility files. This directory contains an <code>NI</code> and a <code>User</code> subdirectory.
Doc	Documentation files
Examples	Example sequences and tests
OperatorInterfaces	LabVIEW, LabWindows/CVI, and Microsoft Visual Basic operator interfaces with source code. This directory contains an <code>NI</code> and a <code>User</code> subdirectory.

Table 1-1. TestStand Subdirectories (Continued)

Directory Name	Contents
Setup	TestStand Installer/Uninstaller
Tutorial	Sequences and code modules that you use in the tutorial sessions in this manual

Learning TestStand

The best way to familiarize yourself with TestStand is to do the following:

1. Thoroughly read the `doc\readme.txt` file distributed with TestStand.
2. Read the remainder of this chapter for an overall idea of the concepts and capabilities of TestStand.
3. Complete the tutorial sessions in Chapters 2–12 as outlined in this manual.
4. Read Chapter 1, *TestStand Architecture Overview*, of the *TestStand User Manual*, and familiarize yourself with the other chapters in that manual.

Beginners should complete the tutorials in this manual. The *TestStand User Manual* generally assumes familiarity with *Getting Started with TestStand*. However, it is still useful to make quick references to other manuals and the online help as questions arise while you learn and use TestStand.

When you begin the tutorial in Chapter 2, *Loading and Running Sequences*, of this manual, you will learn about the windows, menus, commands, and dialog boxes in TestStand. The *TestStand User Manual* contains a chapter devoted to each of the windows and components in TestStand. Scanning these chapters as each topic comes up in *Getting Started with TestStand* will help you find the answers to any questions you might have. The table of contents and index of each manual lists the location of helpful information in that manual.

The tutorial begins with a general introduction to the TestStand sequence editor and continues with sections devoted to building sequences in TestStand. Because each step of the tutorial builds on previous elements, you should follow the outline as given and not skip ahead.

TestStand System Overview

TestStand is a flexible, powerful test executive framework that has the following major features:

- Out-of-the-box configuration and components that give you a full-featured test executive that is ready to run.
- Numerous ways for you to modify the out-of-the-box configuration and components or to add new components. These extensibility mechanisms enable you to create the test executive that meets your particular requirements without modifying the TestStand test execution engine. You can upgrade to newer versions of TestStand without losing your customizations.
- Sophisticated sequencing, execution, and debugging capabilities and a powerful sequence editor that is separate from the run-time execution operator interfaces.
- Three separate run-time execution operator interfaces with source code for LabVIEW, LabWindows/CVI, and Microsoft Visual Basic.
- Independence from particular Application Development Environments (ADEs). You can create test modules in a wide variety of ADEs and call preexisting modules or executables. You can create your own run-time execution operator interface in any language that can control ActiveX Automation Servers.
- Conversion of sequence files from the LabVIEW Test Executive Toolkit Version 5.0 or the LabWindows/CVI Test Executive Toolkit Version 2.0 to TestStand.
- Comprehensive ActiveX Application Programming Interface (API) for building multithreaded test executives and other sequencing applications.

The remainder of this chapter discusses the major software components of TestStand as a product.

Major Software Components of TestStand

This section provides an overview of the major software components of TestStand.

Figure 1-1 shows the high-level relationships between elements of the TestStand system architecture.

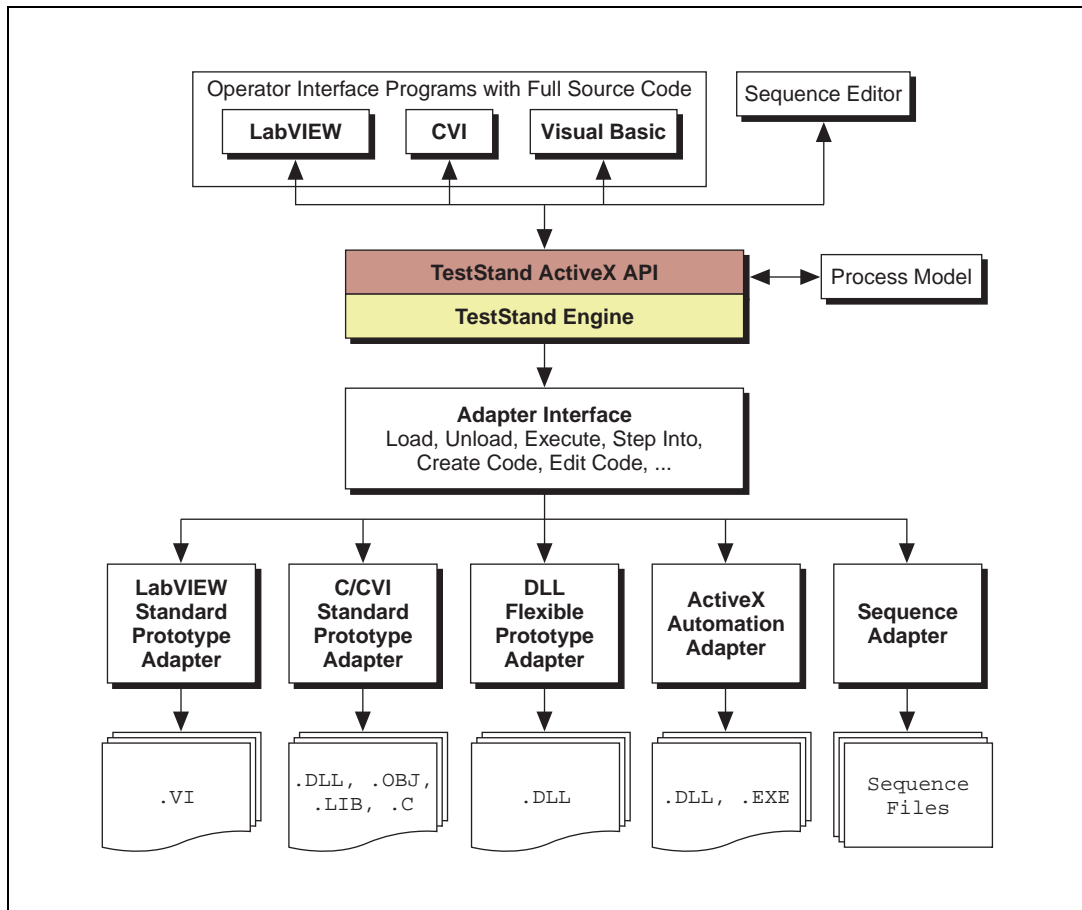


Figure 1-1. TestStand System Architecture

As Figure 1-1 shows, the TestStand engine plays a pivotal role in the TestStand architecture. The TestStand engine can run sequences. Sequences contain steps that can call external code modules. By using module adapters that have a standard adapter interface, the TestStand engine can load and execute different types of code modules. TestStand sequences can call subsequences through the same adapter interface. TestStand uses a special type of sequence called a process model to direct the high-level sequence flow. The TestStand engine exports an ActiveX Automation API that the TestStand sequence editor and run-time operator interfaces use.

TestStand Sequence Editor

The TestStand sequence editor is an application program in which you create, modify, and debug sequences. The sequence editor gives you easy access to all of the powerful TestStand features, such as step types and process models. The sequence editor has the debugging tools that you are familiar with in application development environments such as LabVIEW, LabWindows/CVI, and Microsoft Visual C/C++. These include breakpoints, single-stepping, stepping into or over function calls, tracing, a variable display, and a Watch window.

In the TestStand sequence editor, you can start multiple concurrent executions. You can execute multiple instances of the same sequence, and you can execute different sequences at the same time. Each execution instance has its own Execution window. In trace mode, the Execution window displays the steps in the currently executing sequence. When execution is suspended, the Execution window displays the next step to execute and provides single-stepping options.

TestStand Run-Time Operator Interfaces

Your TestStand software includes three run-time operator interfaces in both source and executable form. Each run-time operator interface is a separate application program. The operator interfaces differ primarily based on the language and ADE in which each is developed. TestStand ships with run-time operator interfaces developed in LabVIEW, LabWindows/CVI and Visual Basic.

Although you can use the TestStand sequence editor at a production station, the TestStand run-time operator interfaces are simpler and are fully customizable. Like the sequence editor, the run-time operator interfaces allow you start multiple concurrent executions, set breakpoints, and single-step. Unlike the sequence editor, however, the run-time operator interfaces do not allow you to modify sequences, and they do not display sequence variables, sequence parameters, step properties, and so on.

Refer to Chapter 15, *Run-Time Operator Interfaces*, in the *TestStand User Manual* for more information about how to customize a run-time operator interface.

TestStand Test Executive Engine

The TestStand Test Executive Engine is a set of DLLs that export an ActiveX Automation API for creating, editing, executing, and debugging sequences. The TestStand sequence editor and run-time operator interfaces use the engine API. You can call the engine API from any programming environment that supports access to ActiveX Automation servers. Thus, you can call the engine API from test modules, including test modules you write in LabVIEW and LabWindows/CVI.

The documentation for the engine API is available as online help. You can access it through the **Help** menu of the sequence editor or from the TestStand program group.

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 the 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 DLLs, and C functions in source, object, or library modules that you create in LabWindows/CVI or other compilers. Also, TestStand must know the list of parameters that the code module requires.

TestStand uses *module adapters* to obtain this knowledge. TestStand currently provides the following module adapters for the following purposes:

- **DLL Flexible Prototype Adapter**—Call C functions in a DLL with a variety of parameter types.
- **LabVIEW Standard Prototype Adapter**—Call any LabVIEW VI that has the TestStand standard G parameter list.
- **C/CVI Standard Prototype Adapter**—Call any C function that has the TestStand standard C parameter list. The function can be in an object file, library file, or DLL. It can also be in a source file that is in the project you are currently using in the LabWindows/CVI development environment.
- **Sequence Adapter**—Call subsequences with parameters.
- **ActiveX Automation Adapter**—Call methods and access the properties of a ActiveX object.

The module adapters contain other important information besides the calling convention and parameter lists. If the module adapter is specific to an Application Development Environment (ADE), the adapter knows how to bring up the ADE, how to create source code for a new code module in the ADE, and how to display the source for an existing code module in the ADE. The DLL Flexible Prototype Adapter can query a DLL type library for the parameter list information and display it to the sequence developer.

Proceed to the next chapter to begin the TestStand tutorials.

Loading and Running Sequences

In this chapter, you learn to load and run sequences in the TestStand sequence editor, and you learn about some of the windows in the sequence editor.

Starting TestStand

To start TestStand, complete the following steps:

1. Launch TestStand by going to the **Start** system menu, and selecting **Programs»National Instruments TestStand»Sequence Editor**. After the sequence editor displays a main window, the Login dialog box appears, as shown in Figure 2-1.

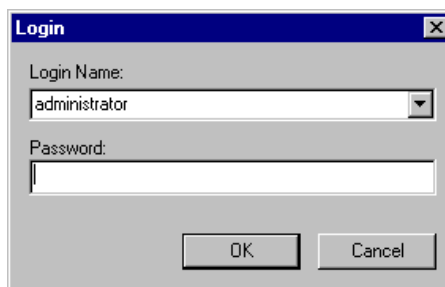


Figure 2-1. Login Dialog Box

2. If the default Login Name is not `administrator` as shown in Figure 2-1, select it by clicking on the Login Name control and selecting `administrator` from the popup list.

3. The default password for the administrator user login is empty, so click on the **OK** button without entering a password. Figure 2-2 shows the main window for the sequence editor.

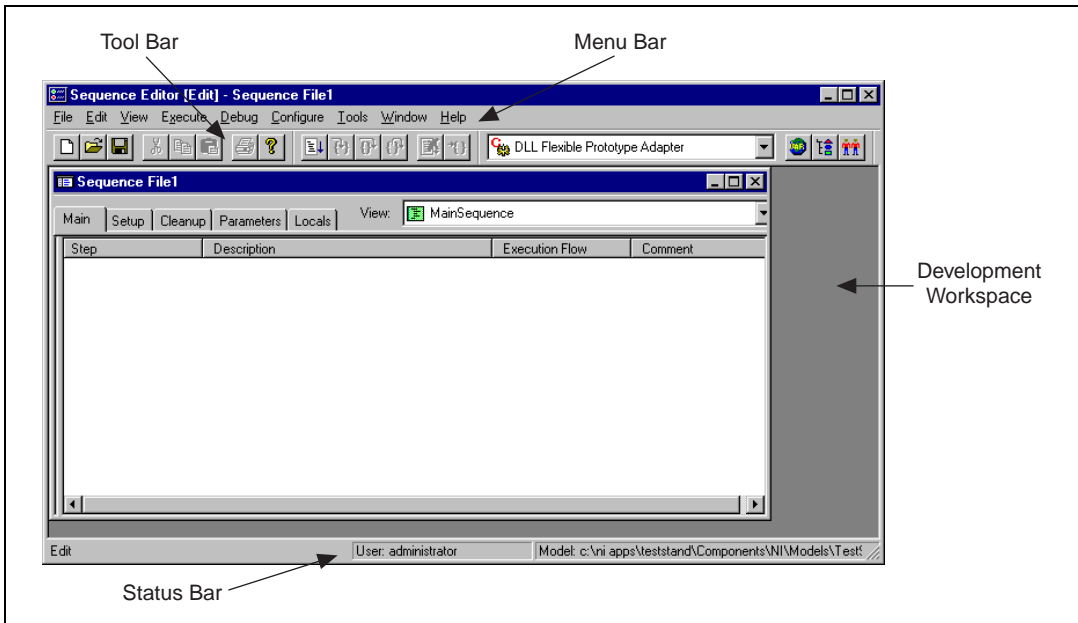


Figure 2-2. Sequence Editor Main Window

Introduction to the Sequence Editor

The sequence editor window has four main parts: the menu bar, the toolbar, the development workspace, and the status bar. A detailed discussion of each of these parts is presented in Chapter 2, *Sequence Editor Concepts*, in the *TestStand User Manual*.

Menu Bar

The menu bar contains the following menus: **File**, **Edit**, **View**, **Execute**, **Debug**, **Configure**, **Tools**, **Windows**, and **Help**. Browse the menus in the sequence editor to familiarize yourself with their contents. Chapter 4, *Sequence Editor Menu Bar*, in the *TestStand User Manual* contains a detailed explanation of each menu item.

Toolbar

The toolbar contains shortcuts to commonly used selections of the menu bar. As shown in Figure 2-3, the toolbar contains three sections: standard, debug, and environment.



Figure 2-3. Sequence Editor Toolbar

- **Standard Section**—Contains buttons for creating, loading, and saving sequence files, cutting and pasting, and printing.
- **Debug Section**—Contains buttons for executing a sequence, stepping into, stepping over, stepping out, pausing, and terminating execution.
- **Environment Section**—Contains the adapter selection ring, and buttons for opening other TestStand station windows.

Development Workspace

The development workspace is the main area of the sequence editor. In this area the sequence editor displays its windows.

Status Bar

The status bar is used to display common information in the sequence editor. As shown in Figure 2-4, the status bar contains four sections: selection help, login display, model display, and keypad indicators.

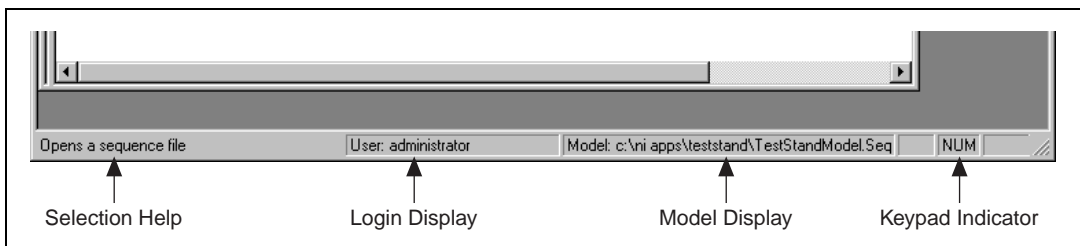


Figure 2-4. Sequence Editor Status Bar

- **Selection Help**—Displays information about the currently selected menu item.
- **Login Display**—Displays the login name of the current user.

- **Model Display**—Shows the pathname of the process model file.
- **Keypad Indicator**—Displays the current state of the <Caps Lock>, <Num Lock>, and <Scroll Lock> keys. You might have to widen the sequence editor to see these indicators.

You can manipulate all windows in TestStand through TestStand menu selections or through the standard means for manipulating windows on the operating system. For example, you can close, maximize, minimize, and position TestStand windows on the screen through any of the Windows standard windowing methods.

Loading a Sequence File

To view some of the features of the TestStand sequence editor, you must first load a sequence file into the TestStand sequence editor. To do this, follow these instructions.

1. Select **File>Open**. When you make this selection, an Open dialog box appears as shown in Figure 2-5.

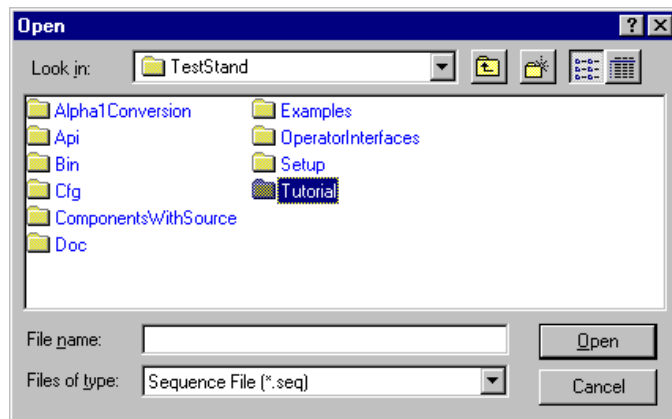


Figure 2-5. Open Dialog Box

2. Navigate to the TestStand\Tutorial subdirectory.
3. Select the Sample1.seq sequence file from the Tutorial subdirectory, and click on the **Open** button.

After you open the sequence file, a new sequence file window appears in the sequence editor, as shown in Figure 2-6.

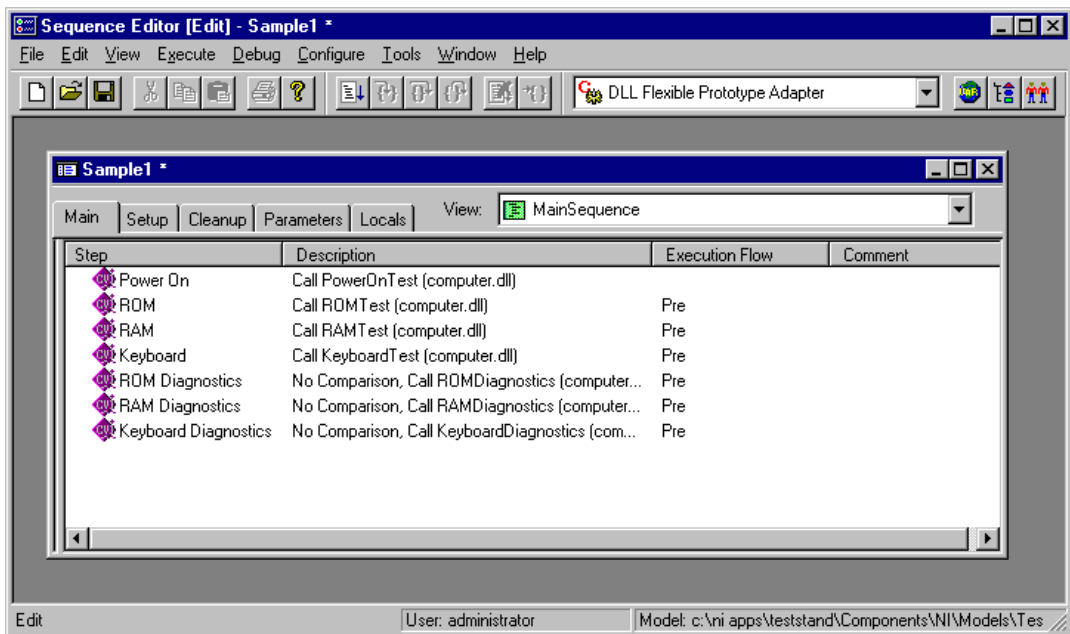


Figure 2-6. Sample1.seq Sequence File Window

Note *If you are not the first person to use this tutorial on your computer, it might be necessary to reinstall TestStand in order to get the unmodified versions of the tutorial files.*

If other people will be using the Getting Started example files on your computer, be sure to use the Save As option to save your files under different file names. When you must save a file, this manual specifies the suggested name.

The Sample1.seq sequence file is a simulated test of your computer in which you can choose various hardware components to “fail” the test. The sequence runs tests that are functions in a dynamic link library (.dll) written with LabWindows/CVI.

The sequence file appears as a separate window within the sequence editor. This window is called a *Sequence File window*. You can load multiple sequence files into the sequence editor, and the sequence editor displays each in its own Sequence File window.

You use the View ring at the top right of the Sequence File window to select the aspect of the file to display. You can use the View ring to view an individual sequence, a list of all sequences in the file, the global variables in the file, or the data and step types that you use in the file.

Figure 2-7 shows the contents of the View ring for the `Sample1.seq` sequence file.

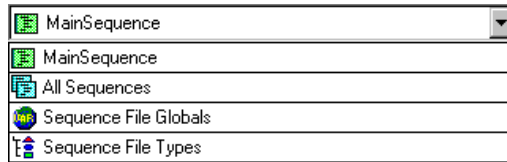


Figure 2-7. Sequence File View Ring

4. Select `MainSequence` in the View ring if it is not already selected.

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

The **Main**, **Setup**, and **Cleanup** tabs each show one of the step groups in the sequence. You can view the contents of each tab by clicking on the tab. Following are the purposes of the steps you insert in each step group:

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

View the contents of each tab and return to the **Main** step group when you finish.

About Sequences

A sequence consists of a series of steps. In TestStand, a step can do many things, such as initializing an instrument, performing a complex test, or making a decision that affects the flow of execution in a sequence. A step can jump to another step, call a subsequence, call an external code module, or change the value of a variable or property.

Sequences can have steps that call other sequences. A sequence can have parameters so you can pass values to it and receive values from it. You define the parameters for a sequence in the **Parameters** tab.

Sequences can have any number of local variables. You can use local variables to hold values that steps use. You can also use local variables for maintaining counts, for holding intermediate values, or for any other value storage. You define the local variables for a sequence in the Locals tab.

Running a Sequence

When you run a sequence in the sequence editor, you are initiating an *execution*. You will examine two methods of running a sequence: running a sequence directly and running a sequence using the *process model*, which is a special type of sequence for directing the high-level sequence flow.

Setting Up Tracing Options

Before you run a sequence, make sure that the sequence editor tracing options are configured properly by completing the following steps.

1. Select **Configure»Station Options**, which displays the Station Options dialog box.

2. Make sure the options on the Execution Tab are set, as shown in Figure 2-8. Update any settings that are different.

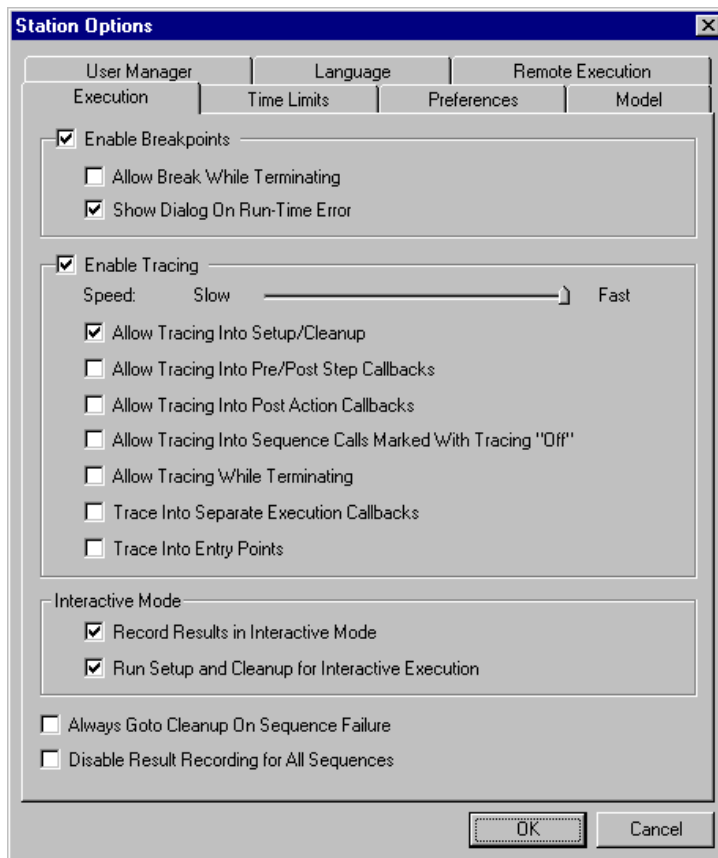


Figure 2-8. Execution Tab on the Station Options Dialog Box

3. Click on **OK** to close the dialog box.

Running a Sequence Directly

The simplest way to start a sequence execution is to run a sequence directly.

Follow these steps to run `MainSequence` in the `Sample1.seq` sequence file window:

1. Select `MainSequence` in the View ring of the sequence file window if it is not already selected.
2. Select **Execute»Run MainSequence**.

When you make this selection, the sequence editor opens a new window. This window is called an *Execution window*. In an Execution window, you can view steps as they execute, the values of variables and properties, and the test report when the execution completes.

Figure 2-9 shows an example Execution window.

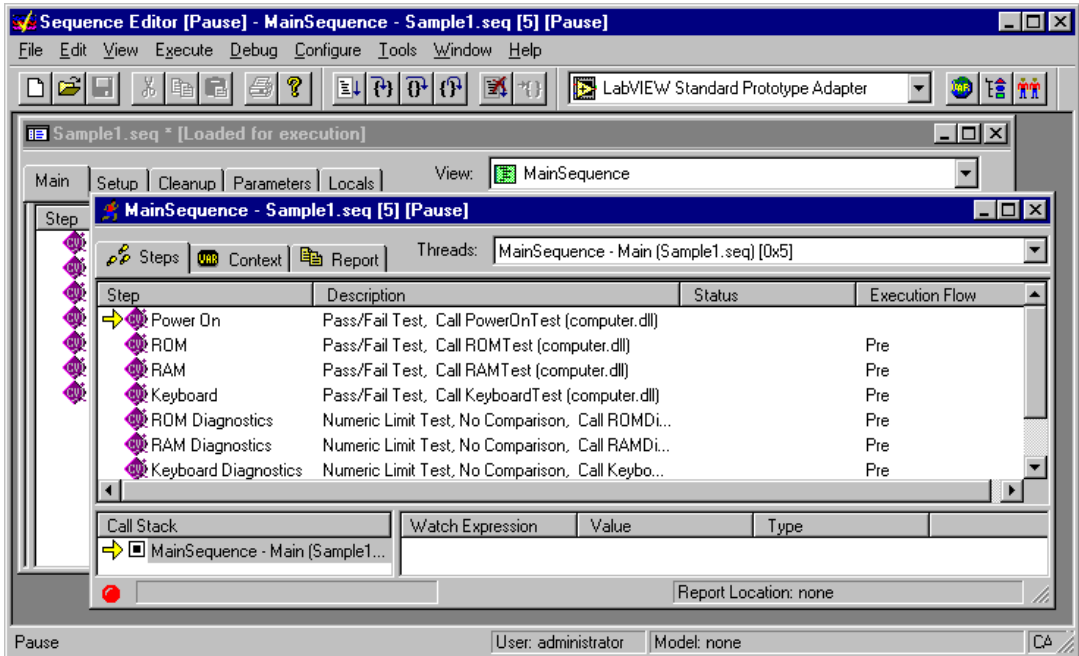


Figure 2-9. Sample1.seq Execution Window

After the execution starts, a Test Simulator dialog box appears in front of the Execution window, as shown in Figure 2-10.

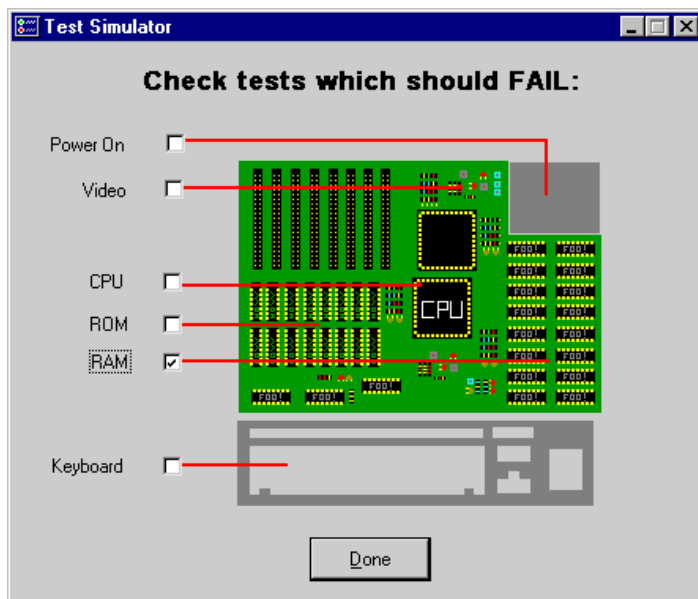


Figure 2-10. Test Simulator Dialog Box

One of the steps in the execution displays this dialog box. The dialog box prompts you to specify which computer component, if any, you want to “fail” during the execution.

3. Select the RAM test by clicking on its checkbox.
4. Click on the **Done** button. Observe the Execution window as it traces through the steps that TestStand runs.

The sequence editor displays the progress of an execution by placing a yellow pointer icon to the left of the currently executing step in the Steps tab. The pointer icon is called the *execution pointer*. Notice that the status column for the RAM test contains the value `Failed`. When the execution completes, the status section of the window title changes from [Running] to [Completed], and the Execution window dims.

5. After the execution completes, close the Execution window by selecting **File>Close**, or by clicking on the X icon on the window title bar.

If you want to rerun the sequence, repeat steps 1 through 5 above.

Note *If you are using a fast computer and you want to slow down the tracing feature of TestStand, you can change the Speed slider control value on the Execution tab of the Station Options dialog box.*

Running a Sequence Using the Process Model

Testing a Unit Under Test (UUT) requires more than just executing a set of tests. Usually, the test executive must perform a series of operations before and after it executes the sequence that performs the 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*. The set of such 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. Because you can modify the process model you can vary the testing process based on your production line, your production site, or the systems and practices of your company.

TestStand provides a mechanism for defining a process model. A process model is in the form of a sequence file. You can edit a process model just as you edit your other sequences. TestStand ships with a fully functional default process model.

A process model defines a set of *entry points*. Each entry point is a sequence in the process model file. By defining multiple entry points in a process model, you give the test station operator different ways to invoke a main sequence.

For example, the default TestStand process model 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 when the active window is a sequence file window and the sequence file contains a sequence called `MainSequence`.

Follow these steps to run `MainSequence` in the `Sample1.seq` sequence file using the `Single Pass` execution entry point.

1. Make sure the `Sample1.seq` sequence file window is the active window.
2. Select **Execute»Single Pass**.

3. Once again, select any test other than the Video Test to fail in the Test Simulator dialog box.
4. Click on **Done** to close the prompt.

Notice that after TestStand executes the steps in the main sequence, the `Single Pass` entry point generates a test report. While TestStand generates the report, a status indicator bar appears at the bottom of the Execution window, as shown in Figure 2-11.

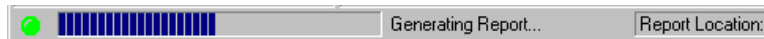


Figure 2-11. Report Generating Status Bar

5. After TestStand generates the report, the Execution window displays the report in the report tab. Examine the test report and notice that it contains information on the results of each step TestStand executes.
You learn more about the test report feature in Chapter 12, [Customizing the Report](#), in this manual.
6. Close the Execution window by selecting the **File>Close** or by clicking on the X icon on the window title bar.
7. Select **Execute>Test UUTs**.
Before executing the steps in the main sequence, the process model sequence displays a UUT Information dialog box requesting a serial number.
8. Enter any number and click on the **OK** button.
9. Select a test to fail in the Test Simulator dialog box.
10. Click on **Done**. Observe the Execution window as the sequence is executing.
After completing the steps in the main sequence, the process model displays a banner that indicates the result of the UUT.
11. Click on the **OK** button to close the UUT Result banner. The process model now generates a report but instead of completing the execution and displaying the report, the process model displays the UUT Information dialog box again.
12. Repeat steps 8 through 11 for several different serial numbers.
13. Click on the **Stop** button to stop the loop and complete the execution.
After the execution completes, TestStand displays a test report for all of the UUTs.

14. Examine the test report and verify that it has indeed recorded the results for each UUT.
15. Close the Execution window by selecting **File»Close** or by clicking on the X icon on the window title bar.

This concludes this tutorial session. In the next session, you learn how to add steps to a sequence and edit step properties.

Editing Steps in a Sequence

In this chapter, you add a step to a sequence and then configure the properties of the step. Also, you add a subsequence call to another sequence.

Setting Up the Example

If you did not directly proceed from Chapter 2, *Loading and Running Sequences*, follow these steps to set up the TestStand sequence editor so you can complete this tutorial session.

1. Close all windows in the sequence editor.
2. Select **File»Open**, open the `Sample1.seq` file in the `Teststand\Tutorial` directory.
3. Display the `MainSequence` sequence in the Sequence File window by selecting it in the View ring.

Adding a New Step

TestStand contains a set of predefined types of steps. Step types define a list of standard properties and behaviors for each step of that type. Step types might call code modules using a module adapter or step types might not use module adapters at all.

The predefined step types available in TestStand include:

- Action
- Numeric Limit Test
- String Limit Test
- Pass/Fail Test
- Label
- Goto
- Statement
- Limit Loader
- Message Popup

- Call Executable
- Call Sequence

For a description of each of these step types, refer to Chapter 10, *Built-In Step Types*, in the *TestStand User Manual*.

In this exercise, you will add a step to the sequence and configure that step to call a function in a DLL code module. Follow these steps to insert a Pass/Fail Test into the sequence:

1. Before you can insert a step that calls a code module, you must specify which module adapter the step uses. You can specify the selected module adapter by clicking on the Adapter Selection ring control on the toolbar as shown in Figure 3-1 or by using the **Configure» Adapters**. The selected adapter applies only to step types that can use any module adapter.

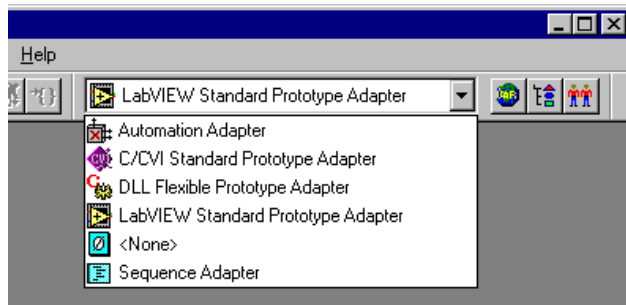


Figure 3-1. Selecting the Module Adapter

When you insert a step in a sequence, TestStand binds the step to the adapter that is currently selected in the ring on the sequence editor toolbar. If you choose <None> for the selected adapter, the step does not call a code module when you insert a step. The icon for the adapter appears as the icon for the step.

Select C/CVI Standard Prototype Adapter in the Adapter Selection Ring. The C/CVI Standard Prototype Adapter allows you to call any C function that has the TestStand standard C parameter list. The function can be in a dynamic linked library (.dll), source file (.c), object file (.obj), or static library file (.lib).

- Right-click on the RAM test in the Sequence File window and select **Insert Step»Tests»Pass/Fail Test** from the menu that appears, as shown in Figure 3-2. This menu is called the *context menu*.

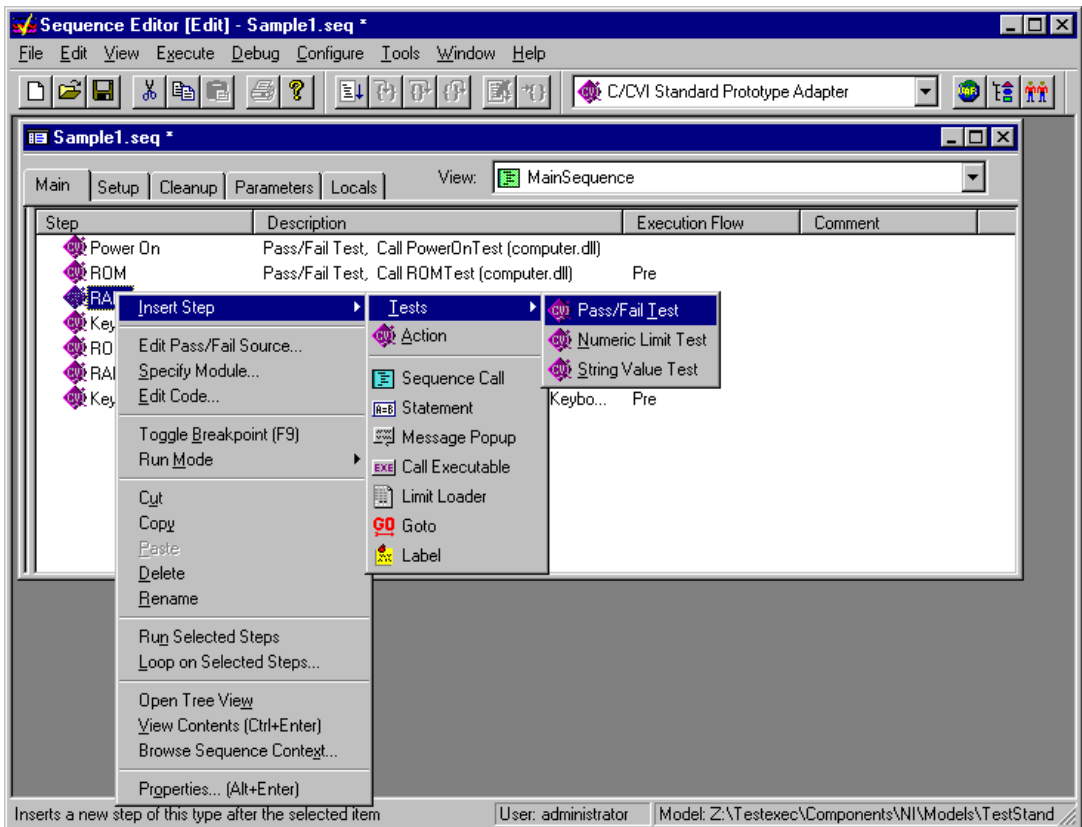


Figure 3-2. Inserting a New Step

When you make this selection, the sequence editor inserts a new Pass/Fail Test step after the RAM step.

Normally, you use a Pass/Fail Test step to call a code module that makes its own pass/fail determination. After the code module executes, the Pass/Fail Test step evaluates a Boolean expression to determine whether the step passes or fails.

- By default, the new test is named Pass/Fail Test. After you insert the step, the name of the step is selected.

4. You can rename the new step by typing `Video Test` and pressing `<Enter>`. If you ever want to rename a step name later, you can right-click on the step name and select the **Rename** command from the context menu.

Specifying the Test Module

After you add a new step to the sequence, you must specify the test module that the step executes.

1. Right-click on the new `Video Test` step and select the **Specify Module** command from the context menu.

When you make this selection, the sequence editor displays a dialog box in which you can specify the code module for the step, along with any parameters to pass when invoking the code module. The actual title of the dialog box varies depending on the module adapter associated with the step.

After you complete the required information in the dialog box, TestStand stores the information as properties of the step. For the C/CVI Standard Prototype Adapter, the sequence editor displays the `Edit C/CVI Module Call` dialog box.

2. Select `Dynamic Link Library (*.dll)` in the `Module Type` ring control. This selection specifies that the code module for the test calls a function within a DLL.
3. Click on the **Browse** button to select the DLL the step will use.
4. Select the `computer.dll` file in the `TestStand\Tutorial` subdirectory. When you select a DLL, TestStand attempts to read the type library of the DLL and lists all the exported functions in the `Function Name` ring control.
5. Select the `VideoTest` function in the `Function Name` ring control by clicking on the arrow to the right of the control. You can click on the scroll bar arrows to scroll down to the `VideoTest` function. This function is a simple routine that returns a Boolean value to indicate whether the test passes or fails.

Figure 3-3 shows the completed dialog box.

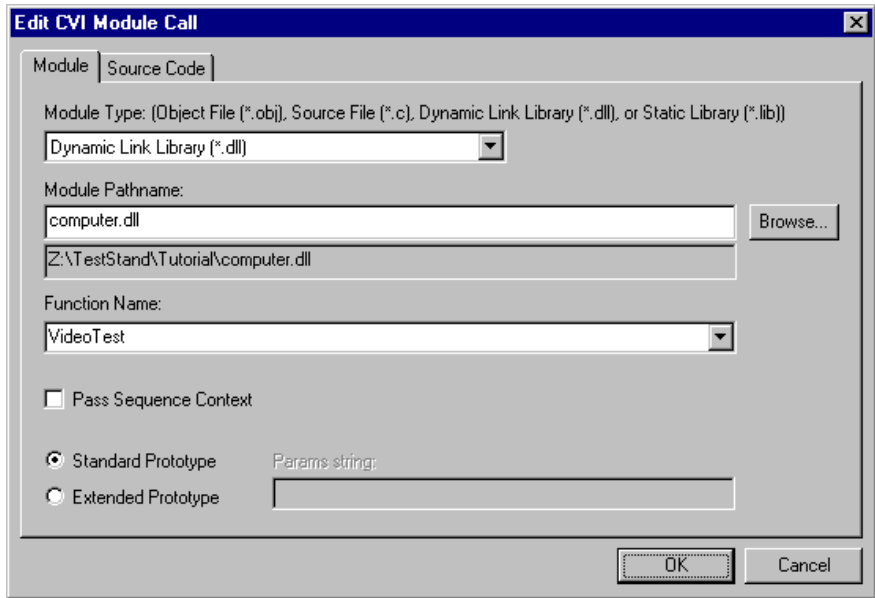


Figure 3-3. Specify Module Dialog Box

6. Click on the **OK** button to close the Edit C/CVI Module Call dialog box and return to the Sequence File window.

Changing Step Properties

Each step in a sequence contains properties. The type of a step determines the set of properties that a step has. All steps have a common set of properties that determine the following:

- When to load the step
- When to execute the step
- What information TestStand examines to determine whether a test passes or fails
- Whether TestStand executes the step in a loop

In this exercise, you will examine some of these common properties and how you can set their values.

1. Right-click on the Video Test and select the **Properties** command from the context menu.

When you make this selection, TestStand displays the Step Properties dialog box for the step, shown in Figure 3-4.

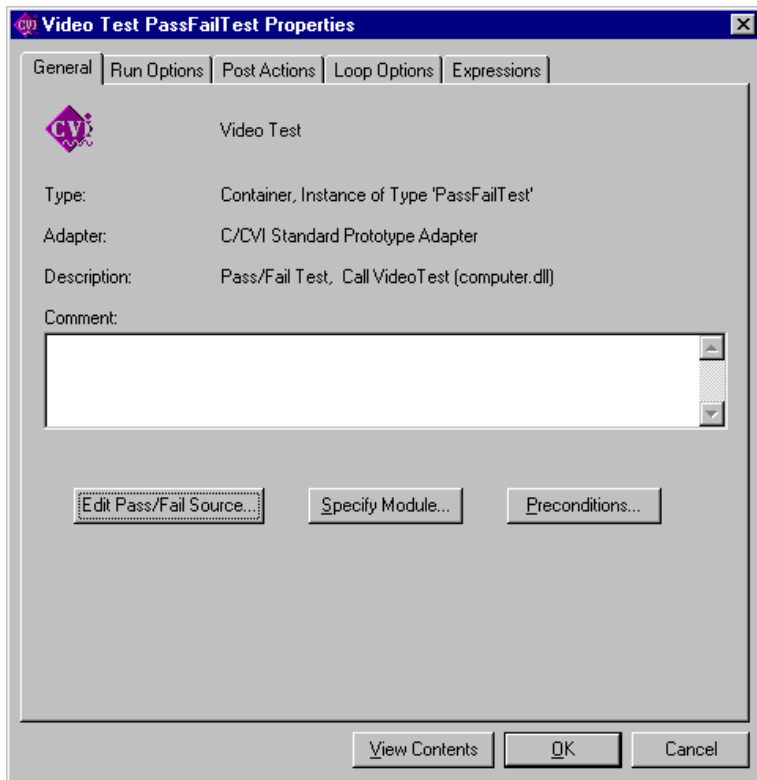


Figure 3-4. Step Properties Dialog Box

2. Click on the **Preconditions** button on the dialog box, which displays the Preconditions dialog box, shown in Figure 3-5.

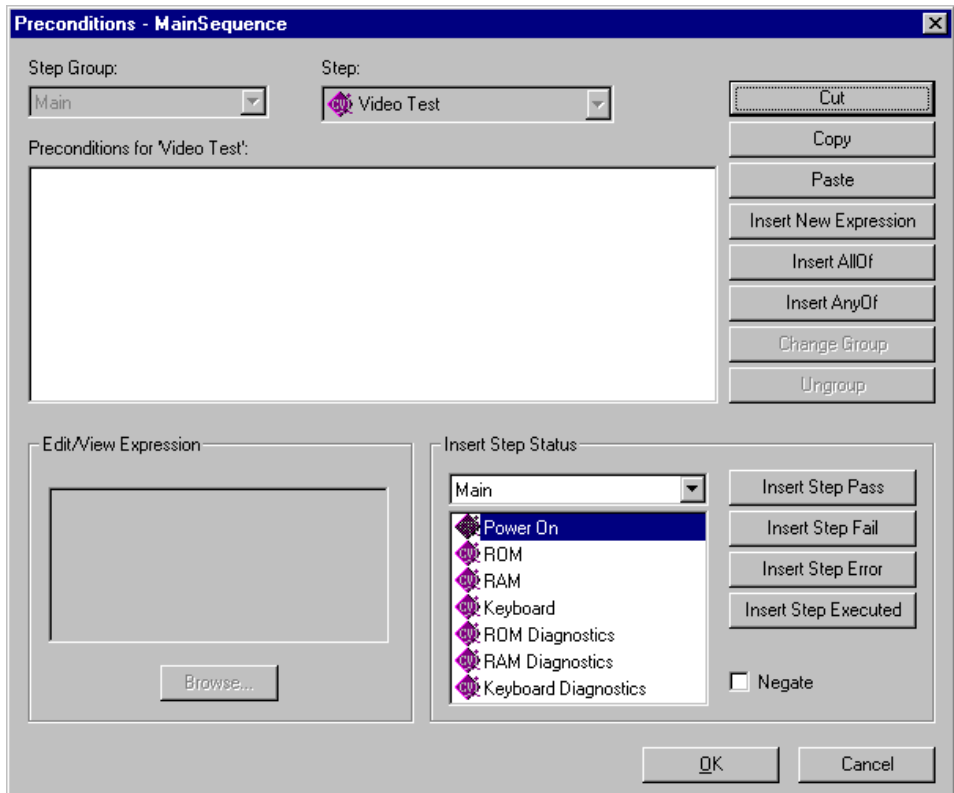


Figure 3-5. Preconditions Dialog Box

A precondition specifies the conditions that must evaluate to 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.

3. For the Video Test, define a precondition so that the step executes only if the Power On step passes, as follows:
 - a. Under the Insert Step Status section of the dialog box, click on the Power On step in the list of step names for the Main step group.
 - b. Add a condition to the precondition list by clicking on the **Insert Step Pass** button. The Preconditions for 'Video Test' text box now contains the string `PASS Power On`, which indicates that the step executes only if the Power On step passes.

- c. Click on the **OK** button to close the Preconditions dialog box and return to the Step Properties dialog box.
4. Click on the Run Options tab to display its tab, shown in Figure 3-6. This tab contains various settings that affect how TestStand runs this step.

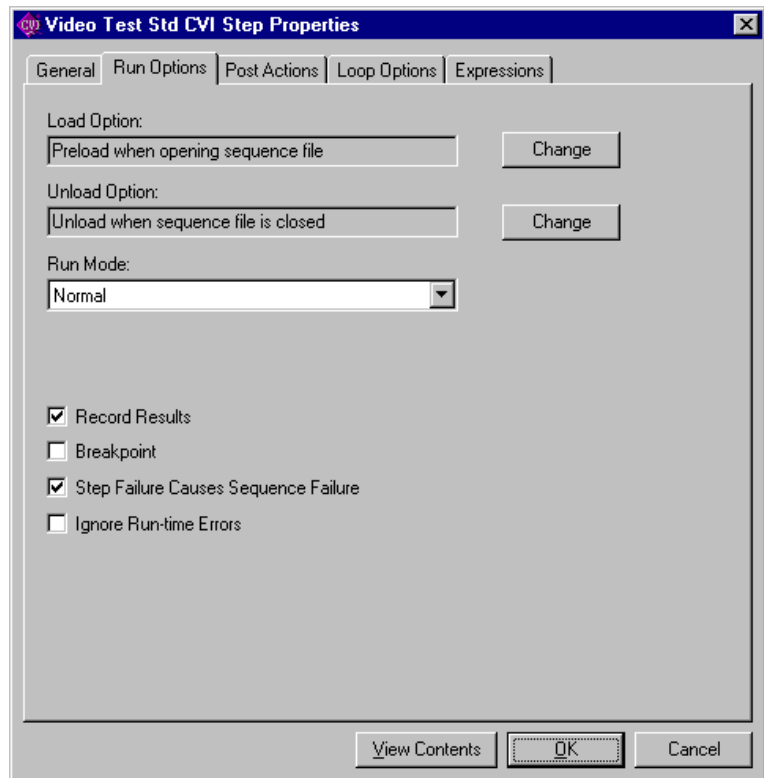


Figure 3-6. Run Options Tab

The Run Options tab on the Step Properties dialog box contains the following controls:

- **Load Option**—You can use this option to specify when a step module is loaded. The available options include:
 - Pre-load when opening sequence file—TestStand loads the step module when it loads the sequence that contains the step into memory.

- Pre-load when execution begins—TestStand loads the step module when any sequence in the sequence file that contains the step begins executing. This value is the default setting.
- Load dynamically—TestStand does not load the step module until the step is ready to call it.
- **Unload Option**—You can use this control to specify when to unload a step module.
 - Unload when precondition fails—TestStand unloads the step module when the precondition for the step evaluates to False.
 - Unload after step executes—TestStand unloads the step module after the step finishes executing.
 - Unload after sequence executes—TestStand unloads the step module after the sequence that contains it finishes executing.
 - Unload when sequence file is closed—TestStand simultaneously unloads the step module and the sequence file that contains the step from memory. This value is the default setting.
- **Run Mode**—You use this ring to set the run mode for the step. The Normal option executes the code module normally. The other options do not execute the code module, but change the step status to `Passed`, `Failed`, or `Skipped`.
- **Record Results**—This option determines whether the step results are added to the result pool for the execution.
- **Breakpoint**—Use this option to break at this step before executing it.
- **Step Failure Causes Sequence Failure**—This option determines whether a sequence status is set to `Failed` when a step fails.
- **Ignore Run-time Errors**—This option prevents the step from reporting a run-time error to the sequence.

5. Click on the Post Actions tab, shown in Figure 3-7. The Post Actions tab specifies an action that occurs after the step executes.

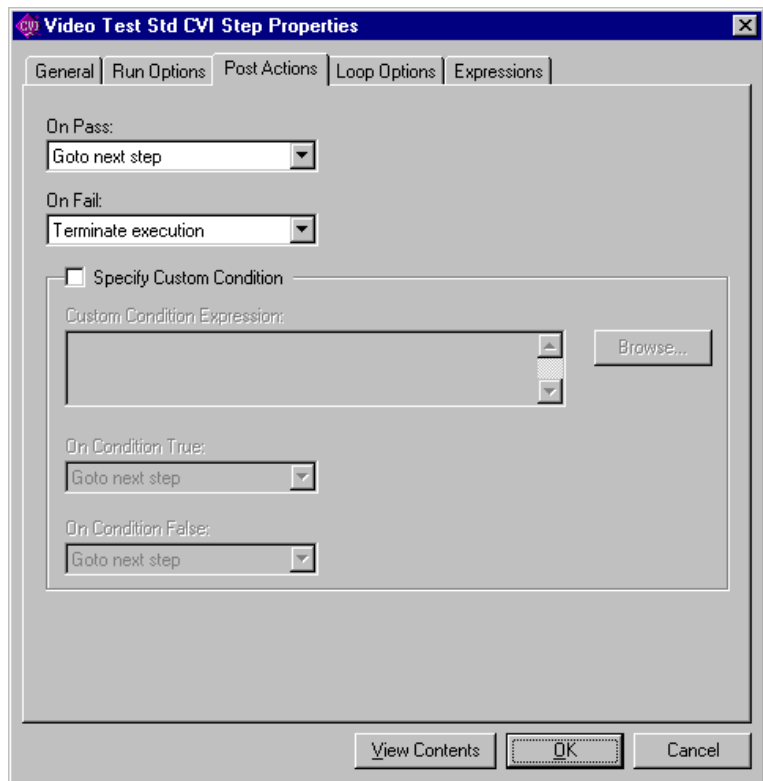


Figure 3-7. Post Actions Tab

You can make the action conditional on the pass/fail status of the step or on any custom condition expression. For example, you might want to go to a particular step or call a callback sequence if the step fails. By default the On Pass and On Fail actions are to go to the next step.

6. Set the On Fail post action to Terminate execution, which forces the sequence execution to terminate if the step fails.

- Click on the Loop Options tab, shown in Figure 3-8.

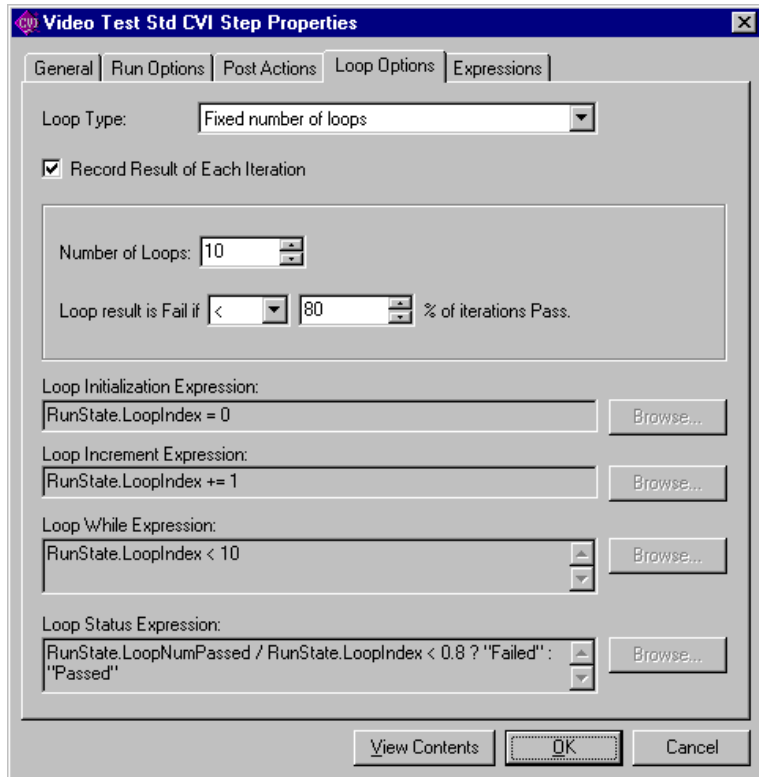


Figure 3-8. Loop Options Tab

You can use the Loop Options tab to configure an individual step to run repeatedly in a loop when it executes. Use the Loop Type ring control to specify the type of looping for the step. Your choices include:

- **None**—TestStand does not loop on the step. This is the default value.
- **Fixed number of loops**—TestStand loops on the step a specific number of times and determines the final pass or fail status of the step based on the percentage of loop iterations in which the step status is `Passed`.
- **Pass/Fail count**—TestStand loops on the step until the step passes or fails a specific number of times or until a maximum number of loop iterations complete. TestStand determines the final status of the step based on whether the specific number of passes or failures occur, or the number of loop iterations reaches the maximum.

- **Custom**—This value allows you to customize the looping behavior for the step. You specify a Loop Initialization expression, a Loop Increment expression, a Loop While expression, and a final Loop Status expression.

8. Change the following control values in the Loop Options tab:

Loop Type Fixed number of loops

Number of Loops 10

Loop result is Fail if < 80 %

With these settings, TestStand executes the Video Test step ten times and sets the overall status for the step to `Failed` if less than eight of the ten iterations pass. Figure 3-8 shows the completed dialog box.

9. Click on the Expressions tab, shown in Figure 3-9.

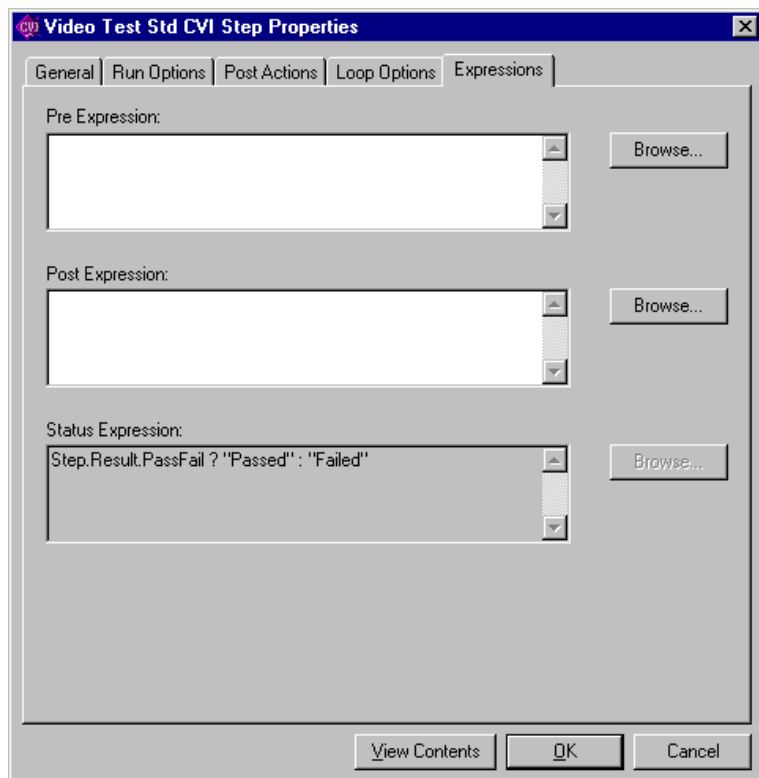


Figure 3-9. Expressions Tab

You can use the Expressions tab to specify expressions that TestStand evaluates before and after TestStand calls the step. You learn about expressions in TestStand later in this manual.

10. Click on the **OK** button to close the Step Properties dialog box.
Notice that the Execution Flow column on the Sequence File window shows that the Video Test contains Post Actions and Loop Options.
11. Select **File»Save As** and save the sequence in the `TestStand\Tutorial` directory as `Sample2.seq`.
12. Run the sequence by selecting **Execute»Single Pass**. Notice that if you select the Video Test to fail, the sequence immediately terminates after calling the Video Test ten times in a loop.
After TestStand generates the report, notice that when the video step executes, the result of each loop iteration is recorded in the report.
13. Close the Execution window by selecting **File»Close** or by clicking on the X icon on the window title bar.

Adding a SubSequence to a Sequence

In TestStand, you can specify another sequence as a step in a calling sequence. You can call a sequence in the current sequence file or a separate sequence file. In this exercise, you will add a sequence call step to your current sequence.

1. Right-click on the Power On step and select the **Insert Step»Sequence Call** command in the context menu. When you make this selection, the sequence editor inserts a Sequence Call step after the Power On step.
2. Rename the step `CPU Test`.
3. Specify which sequence the step invokes, as follows:
 - a. Right-click on the CPU Test step.
 - b. Select the **Specify Module** command from the context menu, which displays the Edit Sequence Call dialog box.
 - c. Click on the **Browse** button to the left of the File Pathname control.

- d. Select the file `SubSequence1.seq` from the `TestStand\Tutorial` directory. Figure 3-10 shows the completed dialog box.

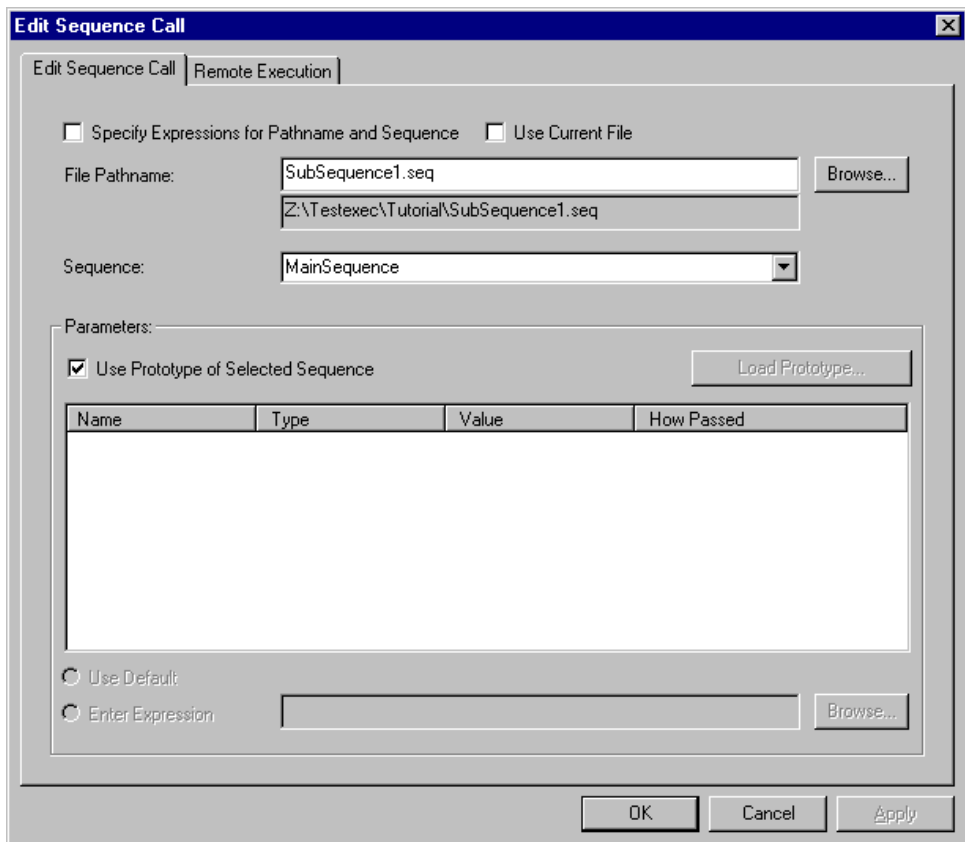


Figure 3-10. Edit Sequence Call Dialog Box

- e. Click on the **OK** button to close the dialog box.
- Select **File»Save** to save your changes to the sequence file.
 - Right-click on the CPU Test step again and select the **Open Sequence** command from the context menu. When you make this selection, the sequence editor opens the `SubSequence1.seq` sequence file and displays the `MainSequence` sequence.

As with all sequence files, you can execute any sequence in the sequence file.

6. Select **Execute»Run MainSequence**. Examine the execution of this sequence.
7. Close the Execution window.
8. Close the SubSequence1 .seq Sequence File window.
9. Select **Execute»Single Pass**.
10. Select a test to fail.
11. Click on **Done**.

After the sequence executes, examine the test report and notice that TestStand logs the results of the steps in the subsequence along with the steps from the parent sequence.

12. Close the Execution window.

This concludes this tutorial session. In the next session, you learn how to use the executing and debugging tools available in TestStand.

Debugging Sequences

In this chapter, you use some of the sequence debugging features of TestStand to single-step through your sequence during an execution. You will debug the source code in a test later in Chapter 6, *Creating and Debugging Tests*.

Setting Up the Example

If you did not directly proceed from Chapter 3, *Editing Steps in a Sequence*, follow these steps to set up the TestStand sequence editor so you can complete this tutorial session:

1. Close all windows in the sequence editor.
2. Select **File»Open** and open `Sample2.seq` from the `TestStand\Tutorial` directory that you created in Chapter 3, *Editing Steps in a Sequence*.
3. Display the `MainSequence` sequence in the Sequence File window by selecting it in the View ring.

Step Mode Execution

To try step mode execution in TestStand, follow these instructions:

1. Select **Execute»Break At First Step**. You use this command to suspend an execution on the first step that TestStand executes. When enabled, this command has a checkmark beside it in the menu.
2. Click on the Cleanup tab in the sequence window.

Recall that TestStand executes the Cleanup step group after the Main step group executes, and regardless of whether the sequence completes successfully or a run-time error occurs in the sequence. If a Setup or Main step causes a run-time error to occur, the flow of execution stops and jumps to the Cleanup step group.

3. Add a step that displays a message popup to the Cleanup step group to verify this behavior, as follows:
 - a. Right-click in the empty step list of the Cleanup tab and select **Insert Step»Message Popup** from the context menu.

- b. Rename the new step Cleanup Message .
- c. Right-click on the Cleanup Message step and select the **Edit Message Settings** command from the context menu. When you make this selection, the sequence editor displays the Configure Message Box Step dialog box.
- d. Enter the text "Cleanup Message" into the Title Expression string control. Make sure that you enclose the string with double quotation marks ("), which indicates to TestStand that the expression is a literal string.
- e. Enter the text "I am now in the Cleanup Step Group", including the quotation marks, into the Message Expression string control.

Figure 4-1 shows the completed dialog box.

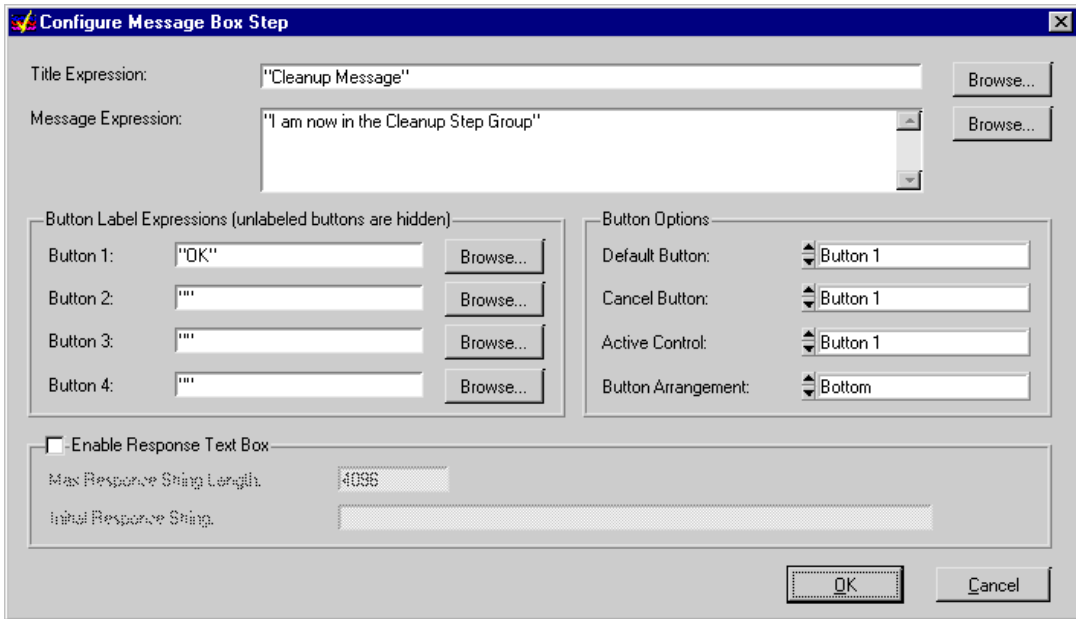


Figure 4-1. Configure Message Box Step Dialog Box

- f. Click on the **OK** button to close the dialog box.
4. Save the sequence by selecting **File»Save As**. Save the sequence as Sample3.seq in the TestStand\Tutorial directory.
5. Execute the sequence directly by selecting **Execute»Run MainSequence**.

After the execution starts, the sequence editor immediately pauses the execution on the first step of the sequence because you previously enabled the Break On First Step option. Figure 4-2 shows the current state of the sequence editor.

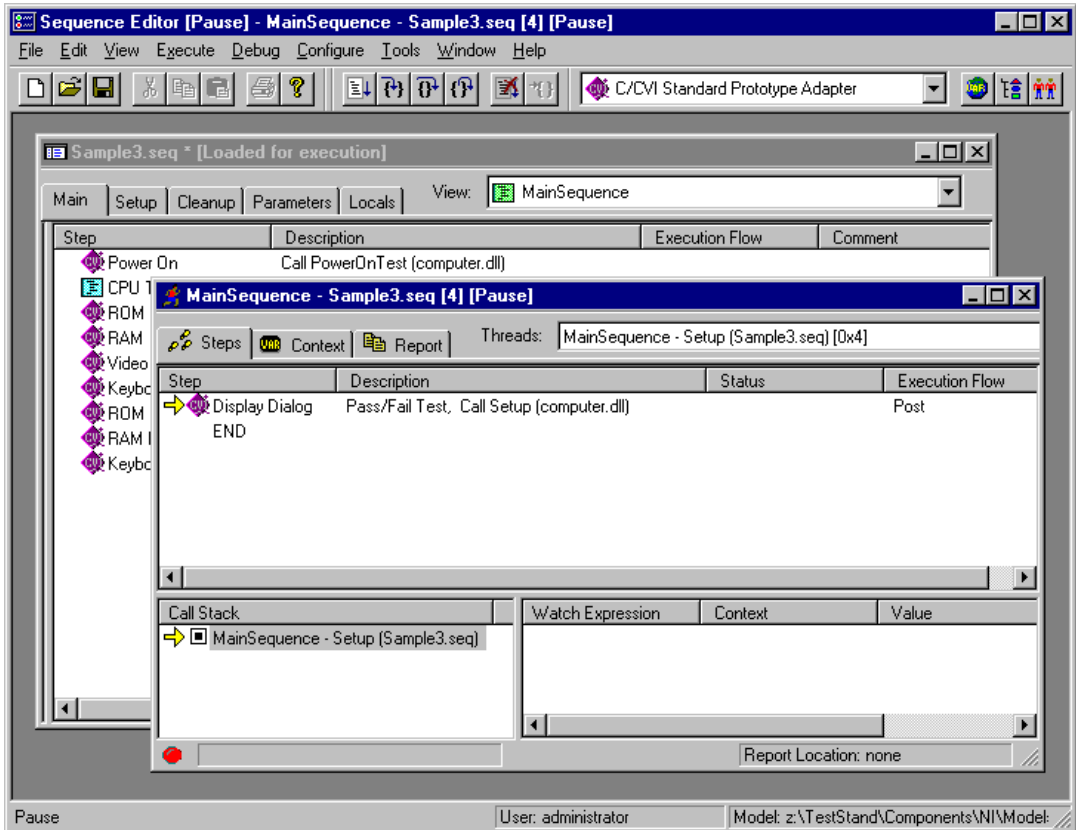


Figure 4-2. Paused Execution of Sample3.seq

Notice that the title of the Execution window contains the running state of the execution, that is, [Pause]. When execution suspends, the Steps tab in the Execution window displays the execution pointer next to the step that will run when execution resumes. The next step that TestStand will execute is the Display Dialog step in the Setup step group.

When execution is in the paused state, you can single-step through the sequence using the **Step Into**, **Step Over**, and **Step Out** commands in the **Debug** menu, or by using the toolbar buttons, as shown in

Figure 4-3. For a detailed discussion of the single-stepping tools, refer to Chapter 6, *Sequence Execution*, in the *TestStand User Manual*.

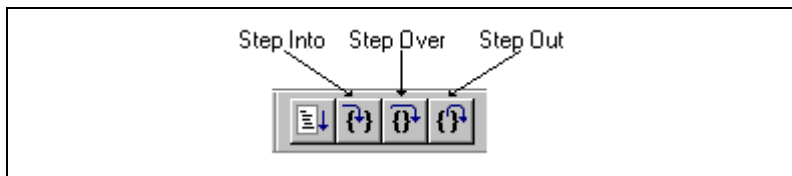


Figure 4-3. Single-Stepping Toolbar Buttons

6. Click on the **Step Over** toolbar button to execute the Display Dialog step. This step displays the Test Simulator dialog box.
7. Select the RAM test to fail, and click on **Done**.
After you close the dialog box, the sequence editor suspends the sequence execution at the end of the Setup step group on END.
8. Activate the `Sample3.seq` Sequence File window by clicking on it or by selecting **Window»Sample3.seq**.
9. Right-click on the CPU Test step in the Main step group tab, and select the **Toggle Breakpoint** command in context menu. Notice that a red stop sign icon appears to the left of the step name.
10. Return to the Execution window by clicking on it or by selecting the window in the **Window** menu. Select **Debug»Resume** to continue the execution. After you make this selection, the sequence editor suspends the execution again on the CPU Test step.
11. Click on the **Step Into** toolbar button to step into the subsequence.

Figure 4-4 shows the Steps view for the Execution window after you step into the subsequence.

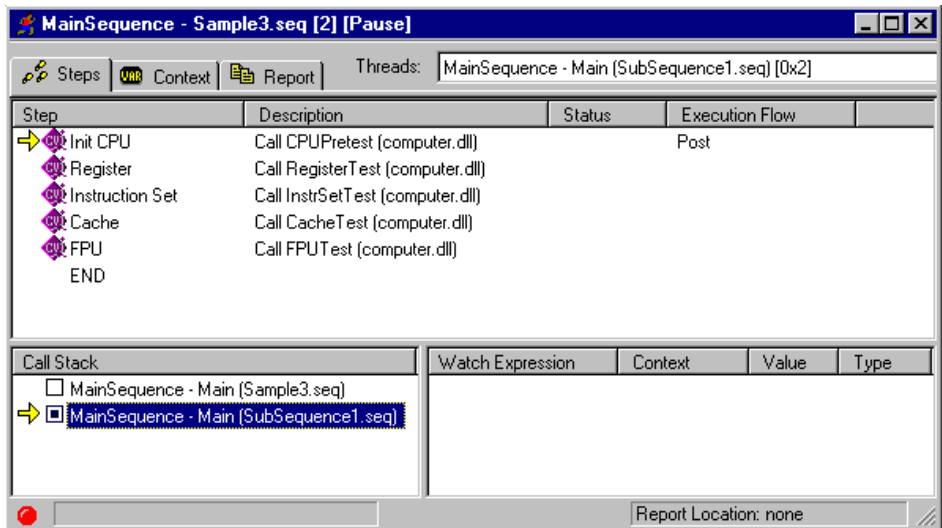


Figure 4-4. Steps View while Suspended in Subsequence

Usually, when a step invokes a subsequence, the sequence that contains the calling step waits for the subsequence to return. The subsequence invocation is *nested* in the invocation of the calling sequence. The chain of active sequences that are waiting for nested subsequences to complete is called the *call stack*. The last item in the call stack is the most nested sequence invocation.

The Call Stack pane in the lower left half of the Execution window displays the call stack for the execution. A yellow pointer icon appears to the left of the most nested sequence invocation. The call stack in Figure 4-4 shows that the main sequence in `Sample3.seq` is calling the main sequence in `SubSequence1.seq`.

When execution suspends, you can view a sequence invocation in the call stack by clicking on its radio button.

12. Click on each radio button in the Call Stack pane to view the status of each sequence invocation.
13. Return to the bottom of the most nested sequence invocation in the call stack.
14. Click on the **Step Over** toolbar button twice to start stepping through the subsequence.

15. Before you reach the end of the sequence, select the **Step Out** toolbar button. TestStand resumes the execution through the end of the current sequence and suspends the execution before the next step after the sequence call or until it reaches a breakpoint, whichever comes first.
16. Continue single-stepping through the sequence using the **Step Over** toolbar button until the execution completes. Notice that the last step executed is the Cleanup Message step that you added to the Cleanup step group.
17. Click on **OK** to close the cleanup message before you complete the execution. The Execution window dims when the execution completes. Do not close the Execution window.
18. Rerun the execution by selecting **Execute»Restart**. The Execution window must be the active window to restart the sequence.
19. After the sequence editor suspends the execution on the first step, select **Debug»Terminate**.
Notice that TestStand still displays the Cleanup Message dialog box even though you terminated the sequence execution. An execution proceeds immediately to the steps in the Cleanup step group when an operator or a run-time error terminates the execution.
20. Click on **OK** to close the cleanup message.
21. Rerun the execution again by selecting **Execute»Restart**.
22. After the sequence editor suspends the execution on the first step, select **Debug»Abort**. Notice that the execution of the sequence immediately stops, and TestStand does not execute any steps in the Cleanup step group.
23. Close the Execution window.
24. Save the sequence by selecting **File»Save**.
25. Close the `Sample3.seq` window.

This concludes this tutorial session. In the next session, you learn how to create and use TestStand variables and properties.

Using Variables and Properties

This chapter teaches you how you can use variables and properties in TestStand, and points out features of TestStand that help you monitor the values of variables and properties.

In TestStand, you can define variables with various scopes to share data between steps of a sequence or even between several sequences. You can define variables that are local to a sequence, variables that are global to a sequence file, and variables that are global to the test station. Use these types of variables as follows:

- You can use *local variables* to store data relevant to the execution of the sequence. Each step and step module can directly access sequence local variables.
- You can use *sequence file global* variables to store data relevant to the entire sequence file. Each sequence and step in the sequence file can directly access these globals.
- You can access *station global* variables from any sequence, step or code module. Unlike other variables, the values of station global variables are saved from one TestStand session to the next. Normally, you use station global variables to maintain statistics or to represent the configuration of your test station.

Setting Up the Example

If you did not directly proceed from Chapter 4, [Debugging Sequences](#), close all windows in the sequence editor so you can complete this tutorial session.

Using TestStand Variables

In this exercise, you learn how to create and use local variables. You can apply the concepts that you learn to sequence file globals and station globals.

1. Open `Sample2.seq` from the `TestStand\Tutorial` directory. `Sample2.seq` is the sequence file you created and saved in Chapter 3, [Editing Steps in a Sequence](#).
2. Display the `MainSequence` sequence in the sequence file window by selecting it in the View ring.
3. Click on the Locals tab of the sequence window. When you make this selection, the view displays all of the local variables currently defined for `MainSequence` in the `Sample2.seq` sequence file. By default, TestStand defines only one variable, `ResultList`, when creating a new sequence. TestStand uses this array variable to store the results from the steps it executes in this sequence.
4. Right-click in the right pane and select **Insert Local»Number** from the context menu, as shown in Figure 5-1. When you make this selection, the sequence editor inserts a new numeric local variable.

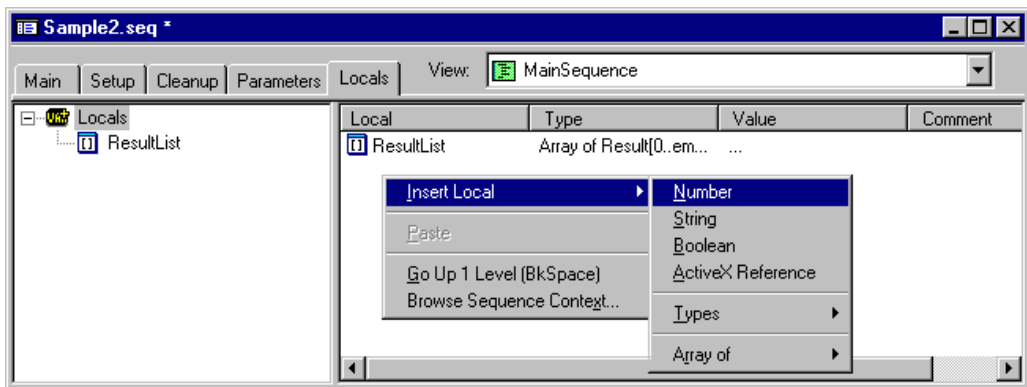


Figure 5-1. Insert Local Context Menu Command

5. Rename the variable `LoopIndex`.

Note *The name of a TestStand variable cannot begin with a number or contain any spaces.*

- 6. Add steps to the sequence to make it loop on a set of steps based on the value of the `LoopIndex` local variable, as follows:
 - a. Click on the Main tab in the sequence file window to display the steps in the Main step group.
 - b. Right-click on the Power On test and select the **Insert Step» Statement** command from the context menu.
 - c. Rename the new step `Reset Loop Index`.

You use Statement steps to execute expressions that TestStand evaluates when it executes the step. For example, you can use a Statement step to increment the value of a local variable in the sequence file.
 - d. Right-click on the `Reset Loop Index` step and select the **Edit Expression** command from the context menu, which displays the Edit Statement Step dialog box, shown in Figure 5-2.

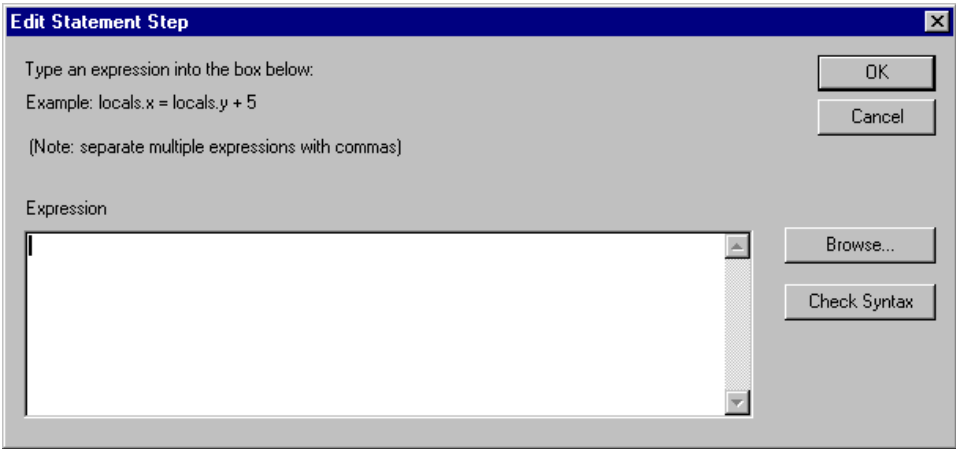


Figure 5-2. Edit Statement Step Dialog Box

- e. Click on the **Browse** button to display the Expression Browser dialog box, shown in Figure 5-3.

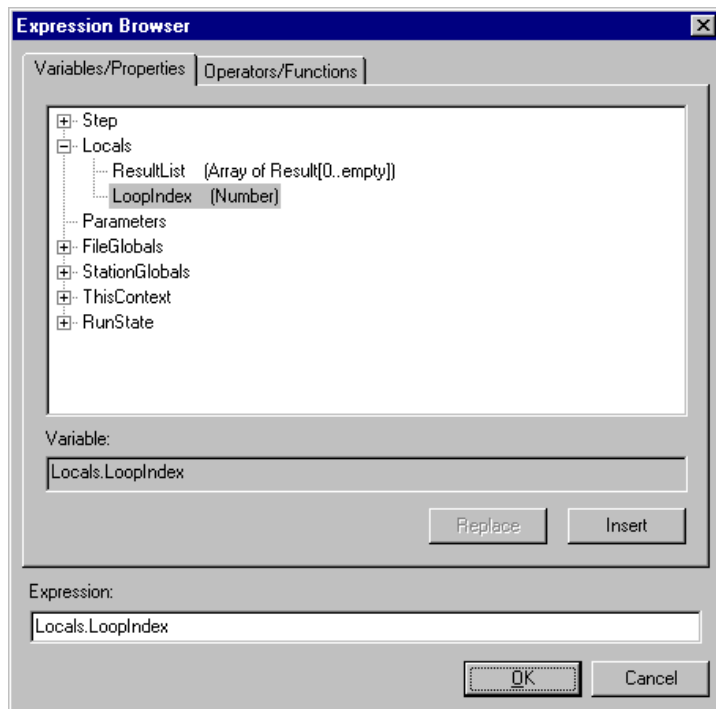


Figure 5-3. Expression Browser Dialog Box

You use the Expression Browser to interactively build an expression. The Expression Browser contains two tabs, Variables/Properties and Operators/Functions. You can select variables and properties from the tree view on the Variables/Properties tab. The Operators/Functions tab contains a list of all predefined operators and functions.

The expression browser has help text for the currently selected operator or function. TestStand supports all applicable expression operators and syntax that you use in C, C++, Java, and Visual Basic.

- f. Expand the Locals item by double clicking on the name or by clicking on the plus icon in front of the item. When you expand a tree view item, the dialog box displays all the items under the base item. Each item in the tree view is a property or a variable of TestStand.

- g. Select the `LoopIndex` variable under the Locals property and click on the **Insert** button. When you make this selection, the expression browser enters `Locals.LoopIndex` into the Expression control.
To refer to a subproperty, you use a period to separate the name of the property from the name of the subproperty. For example, you reference the `LoopIndex` subproperty in the `Locals` property as `Locals.LoopIndex`.
 - h. Click on the Operators/Functions tab, and select the Assignment category from the left pane.
 - i. Select the assignment operator (=) from the right pane.
 - j. Click on **Insert** to add the assignment operator to the expression. You should now see `Locals.LoopIndex =` in the Expression control.
 - k. Place the text cursor directly after the equals sign in the expression control and then type the number zero so that the expression now reads `Locals.LoopIndex = 0`.
 - l. Click on the **OK** button to return to the Edit Statement Step dialog box.
 - m. Click on the **Check Syntax** button to make sure the expression does not contain any illegal syntax.
 - n. Close the Edit Statement Step dialog box by clicking on the **OK** button to return to the Sequence Editor window.
 - o. Right-click on the Reset Loop Index step and select **Insert Step»Label** from the context menu.
 - p. Rename the new step `Loop Begin`. You normally use a Label step as the target for a Goto step, as you will see later in this session. By using a Label step, you can rearrange or delete other steps around the Label step without having to change the target step that the Goto step references.
7. Add a statement step to increment the value of the `LoopIndex` local variable, as follows:
 - a. Right-click on the RAM test and select **Insert Step»Statement** from the context menu.
 - b. Rename the new step `Increment Loop Index`.
 - c. Right-click on the Increment Loop Index step and select **Edit Expression** from the context menu.

- d. Click on the **Browse** button to display the expression browser.
 - e. Use the expression browser to build, or type directly in the Expression control, the following expression:

```
Locals.LoopIndex ++
```

The increment operator (++) is under the Arithmetic group of the Operators/Functions tab.
 - f. Click on **OK** twice to close the both the Expression Browser and the Edit Statement dialog boxes.
8. To complete the loop structure, add a Goto step to the sequence, as follows:
 - a. Right-click on the Increment Loop Index step and select **Insert Step>Goto** from the context menu.
 - b. Rename the step `Loop End`.
 - c. Right-click on the Loop End step and select **Edit Destination** from the context menu.
 - d. Select the Loop Begin step in the Destination control by clicking on the arrow to the right of the control.
 - e. Click on **OK** to close the dialog box.
 9. To complete the loop structure, set a precondition for the Loop End step so that it executes only if the value of the `LoopIndex` variable is below a certain value, as follows:
 - a. Right-click on the Loop End step and select **Properties** from the context menu.
 - b. Click on the **Preconditions** button to open the Preconditions dialog box.
 - c. Click on **Insert New Expression**.
 - d. Click on the **Browse** button in the Edit/View Expression section of the dialog box to open the Expression Browser.
 - e. Using the Expression Browser, create the following expression:

```
Locals.LoopIndex < 5
```

The less than operator (<) is under the Comparison group on the Operators/Functions tab.
 - f. Click on **OK** to close the Expression Browser dialog box.

Figure 5-4 shows the completed Preconditions dialog box.

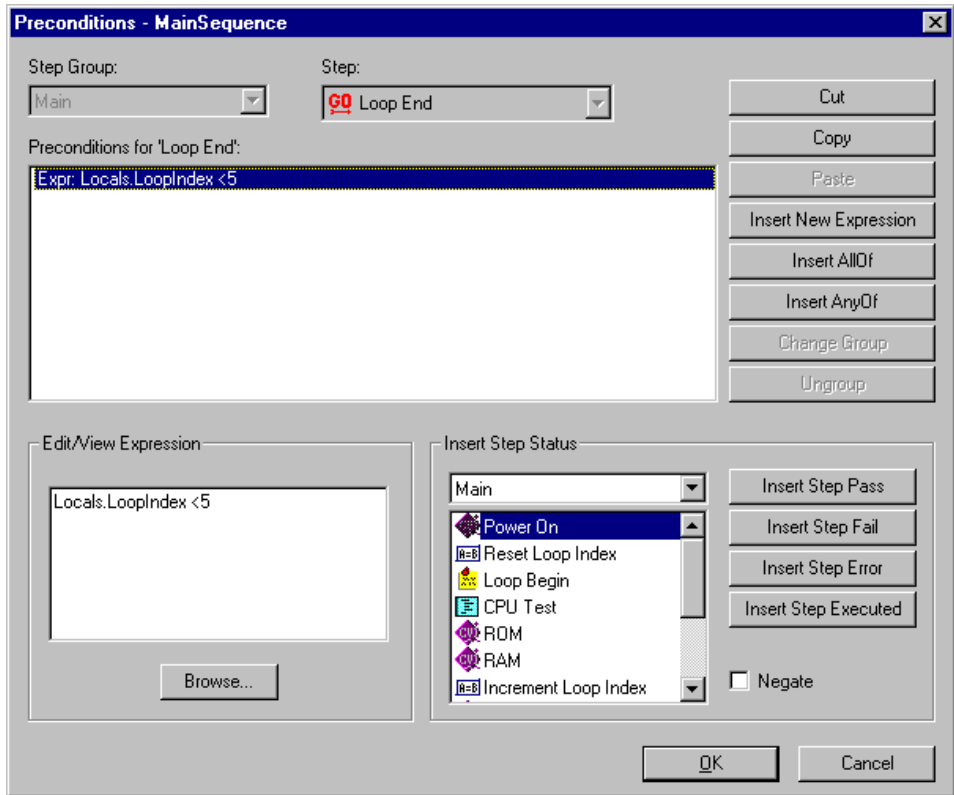


Figure 5-4. Preconditions for the Loop End Step

10. Click on **OK** twice to close the both the Preconditions dialog box and the Step Properties dialog box.
11. Select **File»Save As**. Save the sequence as `Sample4.seq` in the `TestStand\Tutorial` directory.
12. Click on the **Execute** menu to see if the **Break At First Step** option is enabled.
13. If **Break At First Step** is enabled, disable it.
14. Run the Sequence by selecting **Execute»Single Pass**.
15. Click on **Done** in the Test Simulator dialog box.

16. After the sequence executes, examine the test report and notice that TestStand executed the steps within the loop (CPU Test, ROM Test, and RAM Test) five times.
17. Close the Execution window.

Using the Context Tab

In this exercise, you will use the Watch Expression pane of the Execution window to examine the value of the `LoopIndex` variable while TestStand executes the sequence.

1. Right-click on the Loop End goto step and select **Toggle Breakpoint** from the context menu to set a breakpoint on the step. In the sequence window, a red stop sign icon appears beside the Loop End step indicating this breakpoint.
2. Run the sequence by selecting **Execute»Single Pass**.
3. Click on **Done** in the Test Simulator dialog box. The execution suspends on the Loop End step.
4. Click on the Context tab of the Execution window.
5. Expand the `Locals` property in the left upper tree view pane.
6. Click on the `LoopIndex` property under the `Locals` property and notice that the value of the numeric is 1, as shown in Figure 5-5.

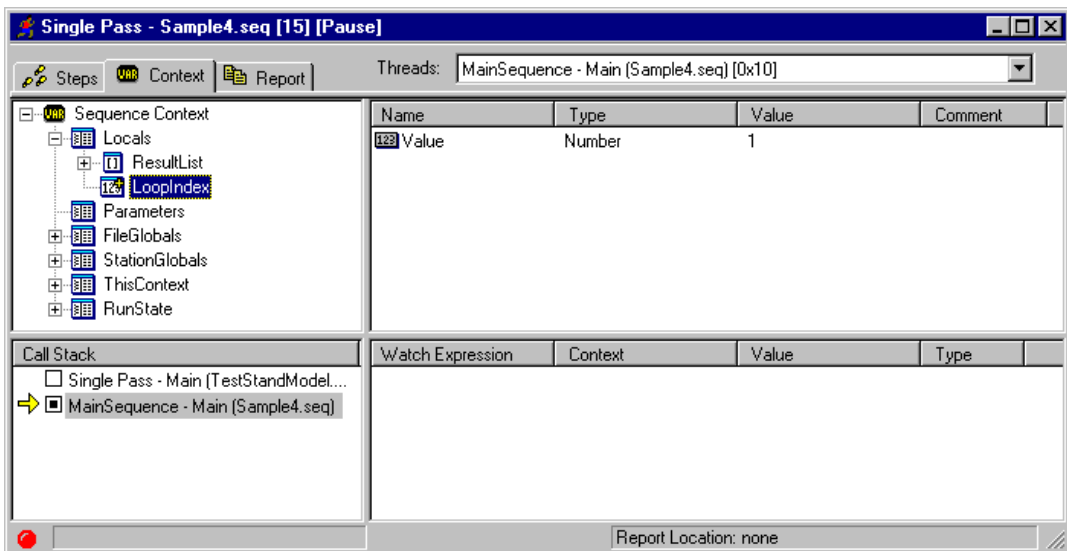


Figure 5-5. Context Tab

The Context tab displays the sequence context for the sequence invocation that is currently selected in the Call Stack pane. The sequence context contains all the variables and properties that the steps in the selected sequence invocation can access. You use the Context tab to examine and modify the values of these variables and properties.

Before executing the steps in a sequence, TestStand creates a run-time copy of the sequence. This allows TestStand to maintain separate local variable and step property values per sequence invocation. TestStand also maintains a *sequence context* that contains references to all global variables and to all local variables and step properties in all active sequences. The contents of the sequence context change depending on the currently executing sequence and step.

You can use the sequence context to access variables and step properties in expressions and through calls to the TestStand ActiveX API from step modules. Refer to the [Using the Watch Expression Pane](#) section later in this chapter for information on expressions. For more information on the TestStand ActiveX API, refer to the *TestStand ActiveX API Reference* online help.

Table 5-1 lists the first-level properties in the sequence context and describes their contents. Refer to Chapter 8, *Sequence Context and Expressions*, in the *TestStand User Manual* for more information on sequence contexts.

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

Sequence Context Subproperty	Description
Step	Contains the properties of the currently executing step in the current sequence invocation. The <code>Step</code> property exists only while a step executes. It does not exist when the execution is between steps, as at a breakpoint.
Locals	Contains the sequence local variables for the current sequence invocation.
Parameters	Contains the sequence parameters for the current sequence invocation.
FileGlobals	Contains the sequence file global variables for the current execution.
StationGlobals	Contains the station global variables for the engine invocation. TestStand maintains a single copy of the station globals in memory.

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

Sequence Context Subproperty	Description
ThisContext	Holds a reference to the current sequence context. You normally use this property to pass the entire sequence context as an argument to a subsequence or a step module.
RunState	Contains properties that describe the state of execution in the sequence invocation, such as the current step, the current sequence, and the calling sequence.

7. Select **Debug»Resume** and notice that the execution resumes and suspends at the Loop End goto step again.
8. Click on the Context tab again and notice that the `LoopIndex` value is now 2. Leave the execution in the Pause state.

Using the Watch Expression Pane

In this exercise, you will monitor the value of the variable `LoopIndex` in the Watch Expression pane. The Watch Expression pane is located in the lower right of the Execution window, as shown in Figure 5-6. The Watch Expression pane displays the values of watch expressions you enter. TestStand updates the values in the Watch Expression pane when execution suspends at a breakpoint. If tracing is enabled, TestStand also updates the values after executing each step.

Normally, you enter watch expressions to monitor the values of variables and properties as you trace or single-step through a sequence. You can drag individual variables or properties from the Context tab to the Watch Expression pane.

To create a watch expression for the variable `LoopIndex`, complete the following steps:

1. Left-click on the `LoopIndex` property in the tree view of the Context tab, and while holding down the mouse button, drag the variable from the tree view to the Watch Expression pane. Release the mouse button when the cursor is over the Watch Expression pane.

Notice that the value of the watch expression already evaluates to 2, as shown in Figure 5-6.

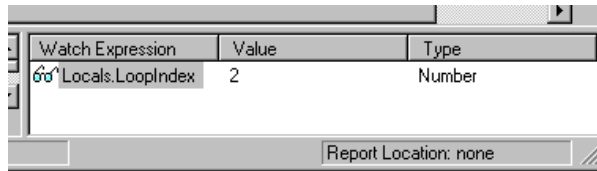


Figure 5-6. Updated Watch Window Pane

2. Now, select **Debug»Resume** and notice that when the execution suspends on the Goto step again, the value of the watch expression changes from 2 to 3.
3. Remove the breakpoint by clicking to the left of the Loop End step icon.
4. Select **Debug»Resume** to complete the execution.
5. Close the Execution window.
6. Save the sequence by selecting the **File»Save**, which saves the changes you made to the `Sample4.seq` sequence file.

This concludes this session of the tutorial. In the next session, you learn how to create and debug tests in the LabVIEW and LabWindows/CVI development environments. For more details on using variables and properties in TestStand, refer to Chapter 5, *Sequence Files*, Chapter 7, *Station Global Variables*, and Chapter 8, *Sequence Context and Expressions*, in the *TestStand User Manual*.

Creating and Debugging Tests

In this chapter, you learn how to create and debug code modules written in LabVIEW and LabWindows/CVI for TestStand.

If you do not use LabVIEW or LabWindows/CVI, you can skip this chapter and proceed to Chapter 7, *Using Run-Time Operator Interfaces*.

Debugging a LabVIEW Standard Prototype Virtual Instrument

In this exercise, you learn how to create LabVIEW test modules that you can use with TestStand, and how to debug them by stepping into a virtual instrument (VI) from the TestStand sequence editor. This session of the tutorial assumes a general familiarity with the LabVIEW development environment. If you are not using LabVIEW, but you do use LabWindows/CVI, you can skip this section and proceed to the *Debugging a LabWindows/CVI Standard Prototype Module* section in this chapter.

Note *Make sure you are using the appropriate version of LabVIEW with TestStand. Refer to the `readme.txt` file in the `TestStand\Doc` directory for more details.*

Setting Up the Example

If you did not directly proceed from Chapter 5, *Using Variables and Properties*, follow these steps to set up the TestStand sequence editor so you can complete this tutorial session.

1. Close all windows in the sequence editor.
2. Select **File»Open** and open `Sample4.seq` from the `TestStand\Tutorial` directory that you created in Chapter 5, *Using Variables and Properties*.

Creating a Virtual Instrument Code Module

In this exercise, you will create a LabVIEW VI to call from a sequence in TestStand.

1. Ensure the LabVIEW Standard Prototype Adapter is properly configured, as follows:
 - a. Select **Configure»Adapters**, which displays the Adapter Configuration dialog box, shown in Figure 6-1.

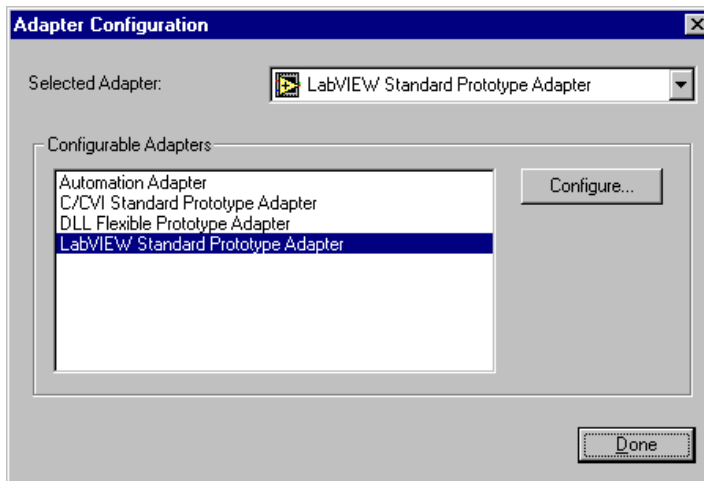


Figure 6-1. Adapter Configuration

- b. Select the LabVIEW Standard Prototype Adapter in the Configurable Adapters section.
 - c. Click on the **Configure** button.

- d. Make sure that the value of the Select Which LabVIEW ActiveX Server to Use control is LabVIEW, as shown in Figure 6-2.

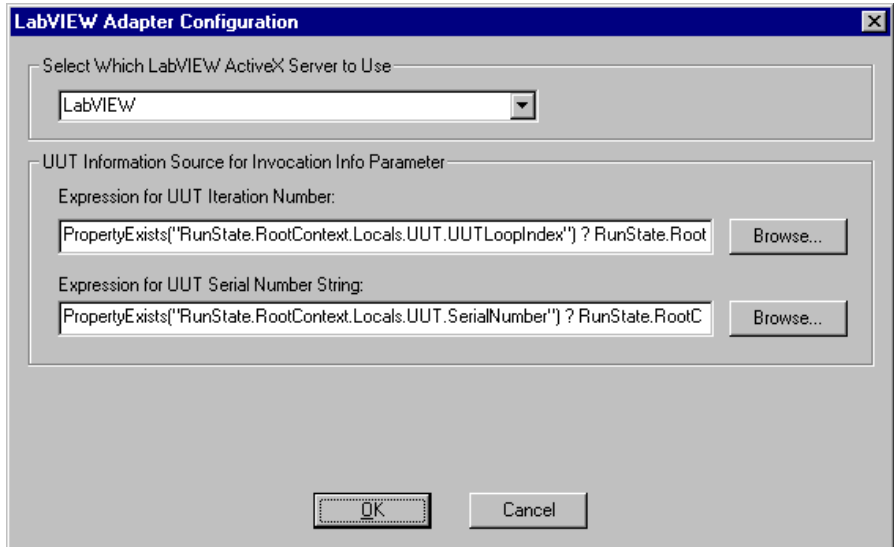


Figure 6-2. LabVIEW Adapter Configuration

- e. Click on **OK**, and then **Done** to close the LabVIEW Adapter Configuration and Adapter Configuration dialog boxes.
2. Make sure the LabVIEW Standard Prototype Adapter is selected in the Adapter Selector Ring control.
 3. Right-click on the Power On test in the Main step group and select **Insert Step»Test»Numeric Limit Test** from the context menu.
 4. Rename the new step Clock Frequency Test.
 5. Right-click on the Clock Frequency Test and select **Specify Module** from the context menu.

6. Enable the Show VI Front Panel When Called check box control. The LabVIEW Step Module Information should now appear as shown in Figure 6-3.

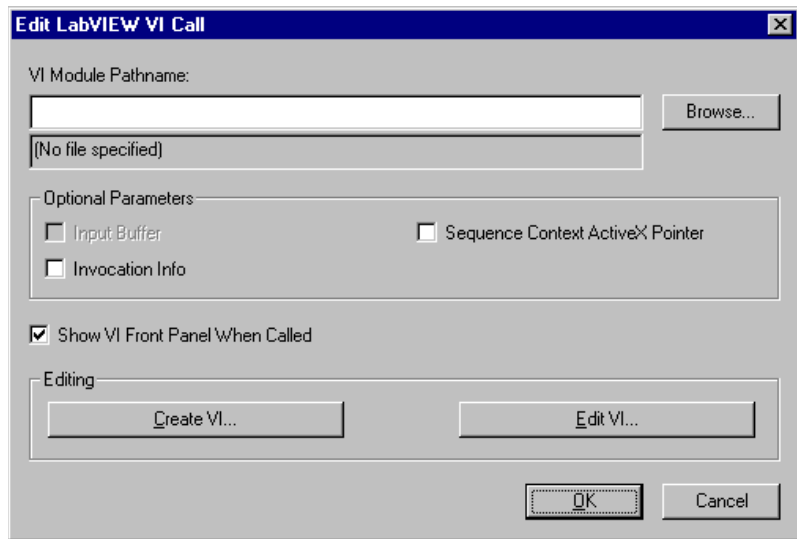


Figure 6-3. LabVIEW Step Module Information

With some module adapters, you can use a source code template to generate the source code shell for a step module. The template files are different for each step type and each module adapter. Multiple source code templates can be available for a particular adapter/step type combination.

For each module adapter that supports source code templates, the Specify Module dialog box contains a button for creating source code. If more than one template is available for the step type, the adapter prompts you to select from a list of available templates. Otherwise, the adapter uses the only available template.

7. Click on the **Create VI** button on the Edit LabVIEW VI Call dialog box. When you make this selection, TestStand prompts you to select a pathname for the step's code module.
8. Find the `TestStand\Tutorial` directory. Type the name `Clock Frequency.vi` in the File name control. The VI might already exist if someone else previously completed this session of the tutorial.
9. Click on **OK** to close the Select a pathname for the step's code module dialog box.

TestStand creates a new VI named `Clock Frequency.vi` using a code template associated with the Numeric Limit Test step type and the LabVIEW Standard Prototype adapter. TestStand then opens the new VI in LabVIEW, as shown in Figure 6-4.

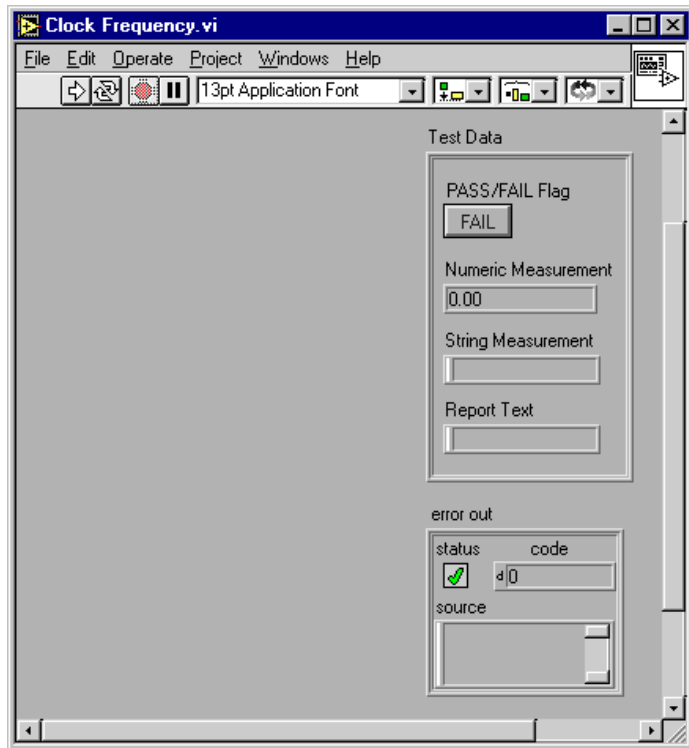


Figure 6-4. New Clock Frequency VI in LabVIEW

Notice that the `Clock Frequency.vi` front panel contains two indicators, `Test Data` and `error out`. The LabVIEW Standard Prototype adapter uses these special data clusters to pass common data between TestStand and the test VI. Although there are several other data clusters that the module adapter supports for passing data `Test Data` and `error out` are required controls. Following is a list of the different elements within these two clusters and how the adapter uses them:

Test Data

- **Pass/Fail Flag**—The test VI sets this Boolean to indicate whether the test passed.
- **Numeric Measurement**—Numeric measurement that the test VI returns.

- **String Measurement**—String value that the test function returns.
- **Report Text**—Output message to display in the report.

error out

- **Status**—The test VI must set this Boolean to `True` if an error occurs.
- **Code**—The test VI can set this to a non-zero value if an error occurs.
- **Source**—The test VI can set this to a descriptive string if an error occurs.

Refer to Chapter 12, *Module Adapters*, in the *TestStand User Manual* for more details on these structures.

10. Add the following LabVIEW controls to the front panel, as shown in Figure 6-5:

- Numeric control with the label Frequency Measurement
- String control with the label Additional Report Text
- Dialog button with the label Return

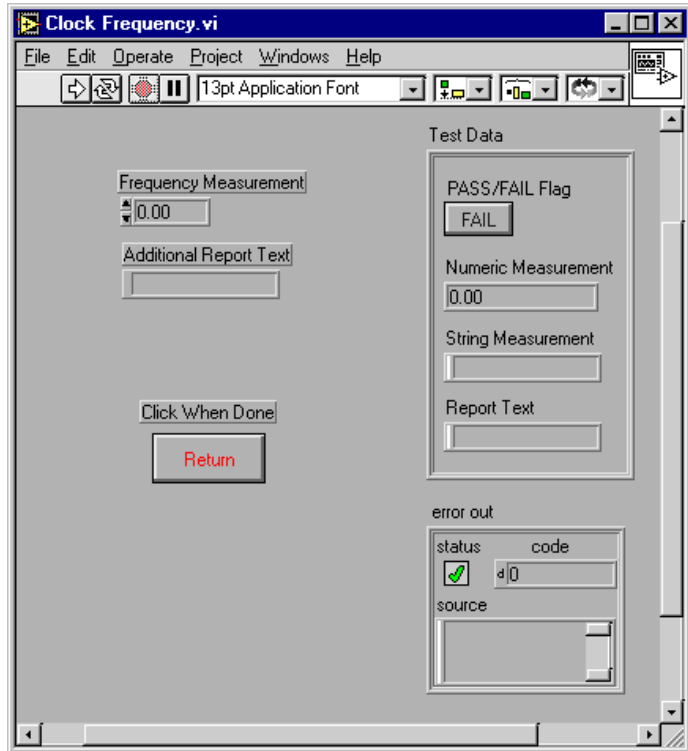


Figure 6-5. Completed Clock Frequency.vi Front Panel

11. Wire the VI block diagram as shown in Figure 6-6.

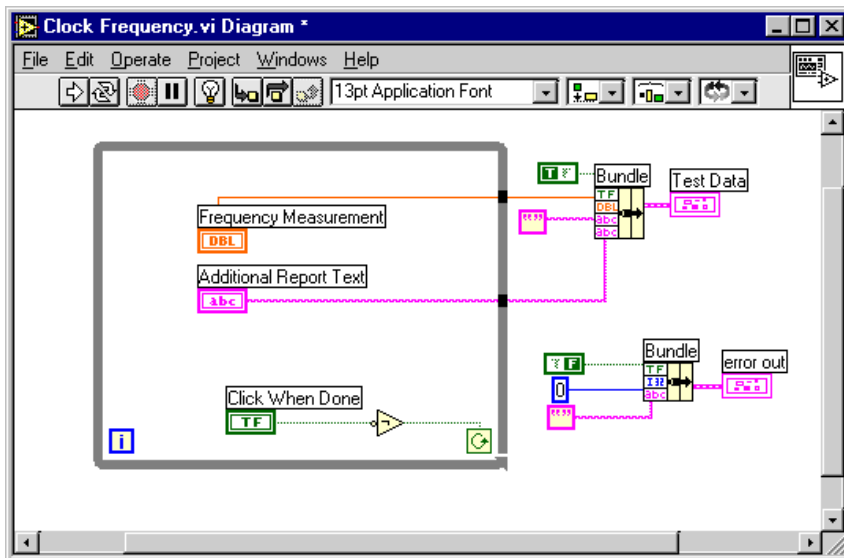


Figure 6-6. Clock Frequency.vi Block Diagram

When you run the VI, the VI loops until you enter values in the Frequency and Additional Report Text controls and click on the **Return** button.

12. After you finish building the VI, save it by selecting **File»Save** in LabVIEW.
13. Close the VI diagram and front panel.
14. Return to the sequence editor, and close the Edit LabVIEW VI Call dialog box by clicking on **OK**.
15. Right-click on the Clock Frequency Test and select the **Edit Limits** command from the context menu, which displays the Edit Numeric Limit Test dialog box.

16. Set the Comparison Type control to LT (<) and the value to 100, as shown in Figure 6-7.

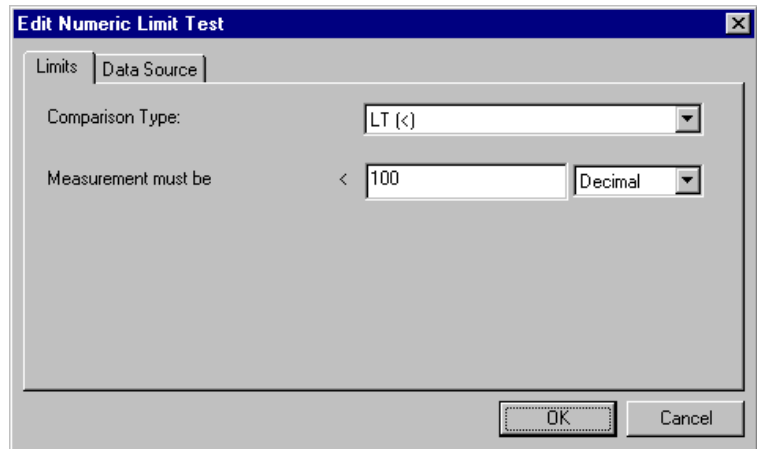


Figure 6-7. Edit Numeric Limits Test Dialog Box

With these settings, TestStand compares the numeric measurement value that the VI returns to the constant value of 100. If the comparison is true, the step passes; otherwise, the step fails.

17. Click on **OK** to close the Edit Numeric Limit Test dialog box.
18. Save the sequence by selecting **File»Save As**. Save the sequence as `Sample5.seq` in the `TestStand\Tutorial` directory.
19. Execute the sequence by selecting **Execute»Single Pass**. When TestStand executes the Clock Frequency Test step, the VI front panel appears and runs the VI.
20. Type a numeric value of 20 in the Frequency Measurement control.
21. Type any text in the Additional Report Text control.
22. Click on the **Return** command button to return from the VI back to the sequence execution.
23. When the sequence completes the execution, examine the test report. Notice the status, measurement, and report text values for the Clock Frequency step.
24. Close the Execution window.

Debugging a Virtual Instrument Code Module

TestStand not only allows you to debug sequences, but also to step into debuggable LabVIEW VIs. In this exercise, you will learn how to debug a LabVIEW test VI while executing a sequence in the sequence editor.

1. Set a breakpoint on the Clock Frequency Test step by right-clicking on the step name and selecting **Toggle Breakpoint**, or by left-clicking to the left of the steps icon.
2. Execute the sequence by selecting **Execute»Single Pass**.
3. Click on **OK** on the Test Simulator prompt. The execution then pauses on the Clock Frequency Test step.
4. Click on the **Step Into** toolbar button, which displays the Clock Frequency.vi front panel in LabVIEW. The VI test is now in a paused state.
5. In LabVIEW, click on the Run toolbar button to execute the VI.
6. Select **Windows»Show Diagram** in the LabVIEW window to show the block diagram of the VI.
7. Click on the **Highlight Execution** toolbar button, shown in Figure 6-8, to highlight the flow of execution within the VI. You can set breakpoints and probes within the VI for more detailed debugging.

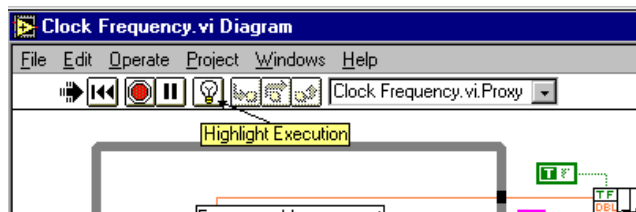


Figure 6-8. LabVIEW Highlight Execution Mode

8. Click on the **Highlight Execution** toolbar button once again to turn off execution highlighting.
9. Return to the front panel by selecting **Windows»Show Panel**.
10. Type a value of 200 in the Frequency Measurement control.
11. Type any text in the Additional Report text control.
12. Click on the **Return** command button to stop the VI.

13. Click on the **Return to Caller** toolbar button to return to the sequence execution with the values in the TestData control and error out indicator.

After the Clock Frequency Test step executes, TestStand suspends the sequence execution on the Reset Loop Index step. Notice that the status of the Clock Frequency Test step is `Failed` as expected.

14. Select **Debug»Resume** to complete the execution.
15. Close the Execution window and the Sequence File window.
16. If TestStand prompts you to save the sequence file, save the sequence as `Sample5.seq` in the `TestStand\Tutorial` directory.

If you develop code modules in LabWindows/CVI, proceed to the next section in this chapter and follow the steps in the *Setting Up the Example* section. Otherwise, go directly to Chapter 7, *Using Run-Time Operator Interfaces*, in this manual.

Debugging a LabWindows/CVI Standard Prototype Module

In this exercise, you learn how to create a LabWindows/CVI code modules, and how to debug the module by stepping into the LabWindows/CVI code from the sequence editor. This session of the tutorial assumes a general familiarity with the LabWindows/CVI development environment. If you are not using LabWindows/CVI, you can skip this section and proceed to Chapter 7, *Using Run-Time Operator Interfaces*.

Note *Make sure you are using the appropriate version of LabWindows/CVI with TestStand. Refer to the `readme.txt` file in the `TestStand\Doc` directory for more details.*

Setting Up the Example

If you did not directly proceed from Chapter 5, *Using Variables and Properties*, follow these steps to set up the TestStand sequence editor so you can complete this tutorial session.

1. Close all windows in the sequence editor.
2. Select **File»Open** and open `Sample4.seq` from the `TestStand\Tutorial` directory that you created in Chapter 5, *Using Variables and Properties*.

Creating a C/CVI Code Module Test

In this exercise, you will create a LabWindows/CVI code module that can prompt for a numeric value from the operator and pass the data back to TestStand.

1. Ensure the LabWindows/CVI Standard Prototype Adapter is properly configured to execute code modules in an external instance of LabWindows/CVI as follows:
 - a. Select **Configure»Adapters**, which displays the Adapter Configuration dialog box, shown in Figure 6-9.

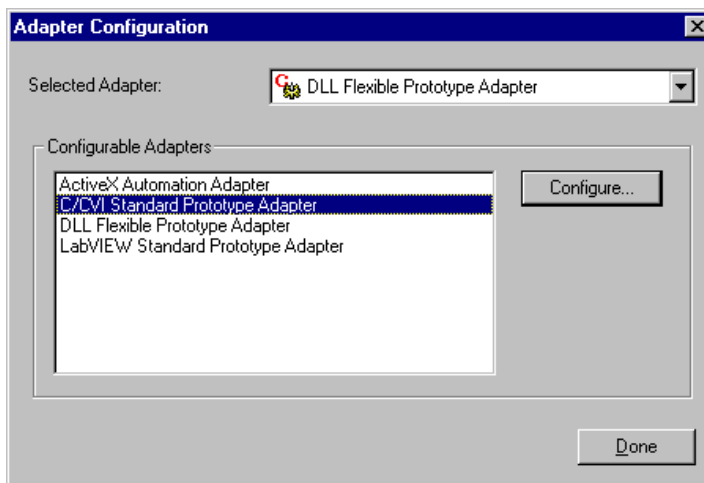


Figure 6-9. Adapter Configuration

- b. Select the LabWindows/CVI Standard Prototype Adapter in the Configurable Adapters section.
 - c. Click on the **Configure** button, which displays the C/CVI Standard Adapter Configuration dialog box.

- d. Make sure that Execute Steps in an External Instance of CVI is enabled, and that the pathname of the LabWindows/CVI project containing the execution server is the file `tscvirun.prj` from the `TestStand\AdapterSupport\CVI` directory, as shown in Figure 6-10.

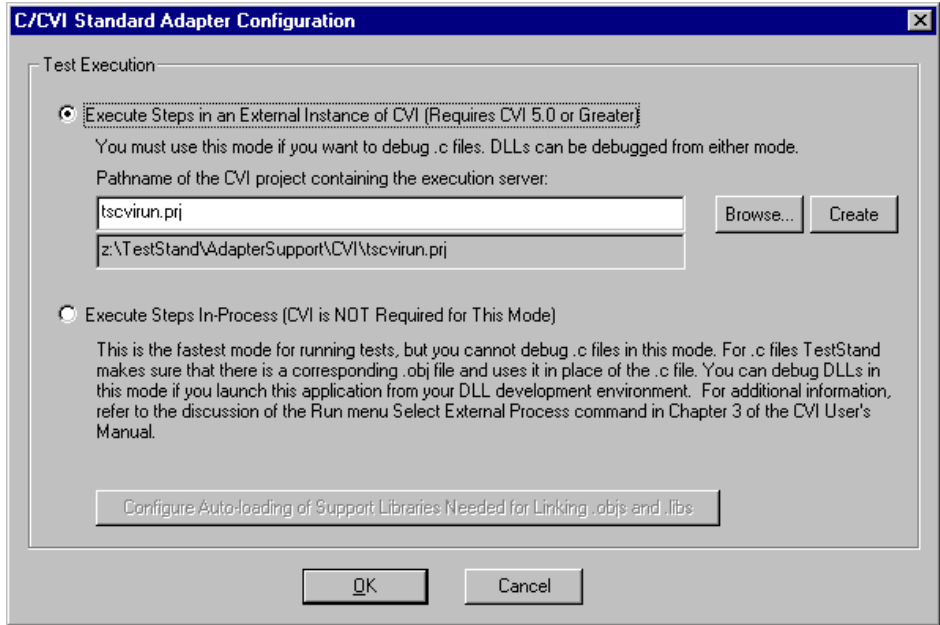


Figure 6-10. C/CVI Standard Adapter Configuration

- e. Click on **OK** to close the C/CVI Standard Adapter Configuration dialog box. TestStand warns you that changing where tests are executed unloads all modules.
 - f. Click on **OK** on the Warning dialog box.
 - g. Click on **Done** to close the Adapter Configuration dialog box.
2. Make sure that C/CVI Standard Prototype Adapter is selected in the Adapter Selector Ring control.
 3. Right-click on the Power On test in the Main step group and select **Insert Step»Tests»Numeric Limit Test** from the context menu.
 4. Rename the step `Clock Frequency Test`.
 5. Right-click on the `Clock Frequency Test` and select **Specify Module** from the context menu. When you make this selection, the sequence editor displays the Edit C/CVI Module Call dialog box.

6. For the Module Type ring control, select Dynamic Link Library (*.dll).
7. Type the name `frequency.dll` in the Module Pathname control. The DLL file may already exist if someone previously completed this session of the tutorial.
8. Enter the name `GetFrequency` in the Function Name control of the Edit C/CVI Module Call dialog box.

Figure 6-11 shows the completed Module tab.

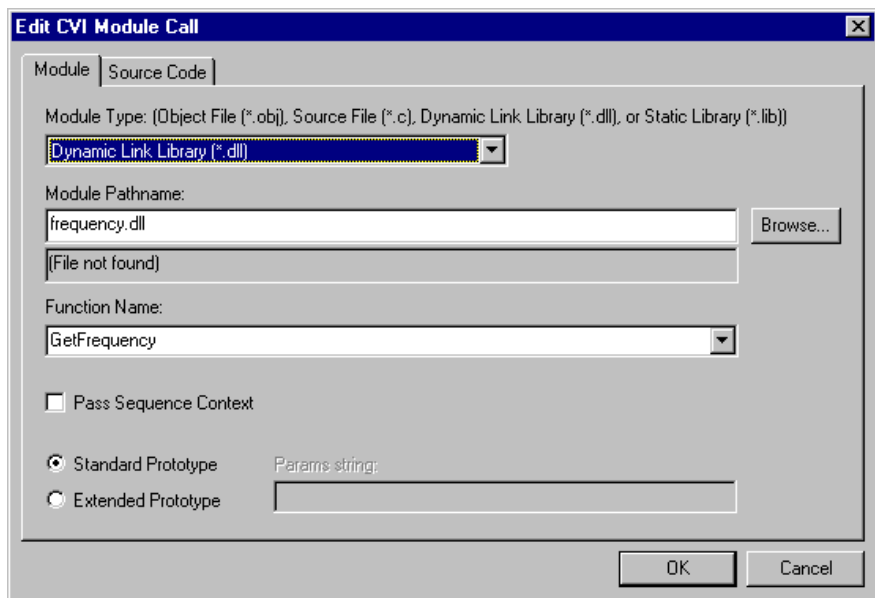


Figure 6-11. Edit CVI Module Call—Module Tab

9. Click on the Source Code tab. You can use the Source Code tab to generate or edit the source code for the function the step calls.
10. Click on the **Browse** button to the right of the Pathname of Source File Containing Function control, and select the `frequency.c` file in the `TestStand\Tutorial` directory.
11. Click on the **OK** button to close the Select a pathname for the source file dialog box.
12. Click on the **Browse** button to the right of the Pathname of CVI Project File to Open control, and select the `frequency.prj` file in the `TestStand\Tutorial` directory.

13. Click on the **OK** button to close the Select a pathname for the CVI project file dialog box.

Figure 6-12 shows the completed Source Code tab for the Edit C/CVI Module Call dialog box.

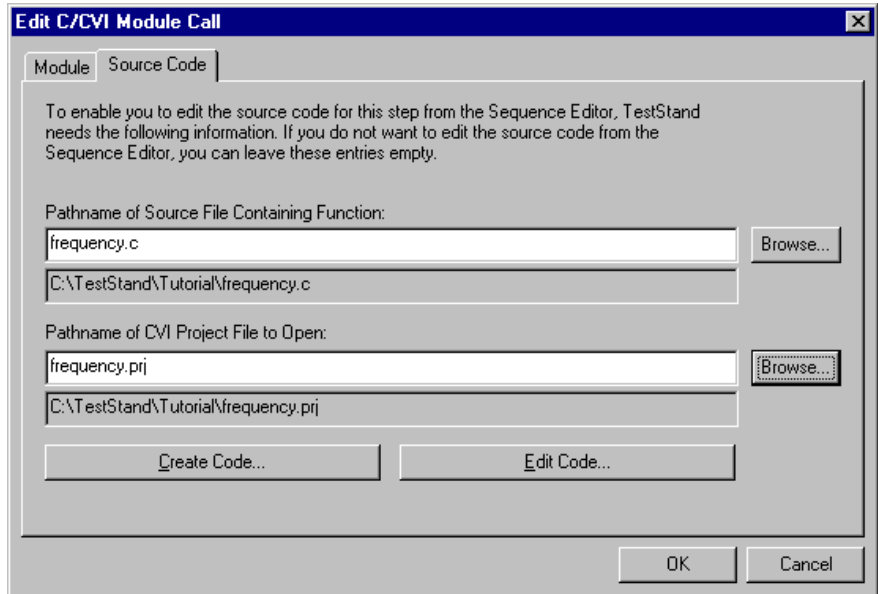


Figure 6-12. Edit CVI Module Call—Source Code Tab

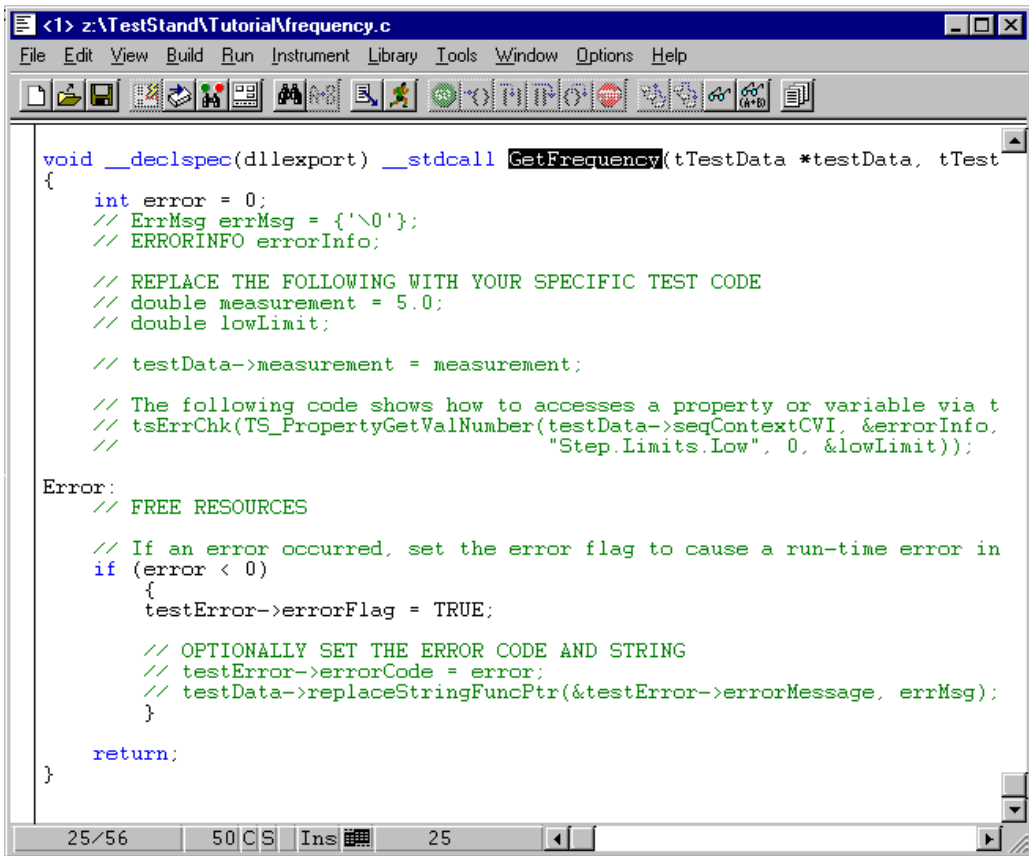
Some module adapters can generate a source code shell for a step module using predefined templates. The available templates vary based on the step type and each module adapter. For each module adapter that supports source code templates, the Specify Module dialog box displays a command button for creating source code. If more than one template is associated with the step type and the specific adapter, the adapter prompts you to select a template, otherwise the adapter creates the code module from the default template.

14. Click on the **Create Code** button.

When you make this selection TestStand does the following:

1. Launches an external instance of LabWindows/CVI
2. Creates a new project file if one does not exist
3. Loads the project file in LabWindows/CVI
4. Creates a new source file if it does not exist

5. Generates a template function in the source file
 6. LabWindows/CVI highlights the function name
15. Figure 6-13 shows the generated function in the `frequency.c` source file.



```

void __declspec(dllexport) __stdcall GetFrequency(tTestData *testData, tTest
{
    int error = 0;
    // ErrMsg errMsg = {'\0'};
    // ERRORINFO errorInfo;

    // REPLACE THE FOLLOWING WITH YOUR SPECIFIC TEST CODE
    // double measurement = 5.0;
    // double lowLimit;

    // testData->measurement = measurement;

    // The following code shows how to accesses a property or variable via t
    // tsErrChk(TS_PropertyGetValNumber(testData->seqContextCVI, &errorInfo,
    //                                     "Step.Limits.Low", 0, &lowLimit));

Error:
    // FREE RESOURCES

    // If an error occurred, set the error flag to cause a run-time error in
    if (error < 0)
    {
        testError->errorFlag = TRUE;

        // OPTIONALLY SET THE ERROR CODE AND STRING
        // testError->errorCode = error;
        // testData->replaceStringFuncPtr(&testError->errorMessage, errMsg);
    }

    return;
}

```

Figure 6-13. Generated Result from Create Code Command

Note *If someone else previously completed this session of the tutorial, the function `GetFrequency` may already exist in the source file. When LabWindows/CVI prompts you to replace the existing function, click on the **Replace** button to continue with this tutorial session.*

The `GetFrequency` prototype function contains two parameters, `tTestData` and `tTestError`. The LabWindows/CVI Standard Prototype Adapter uses these parameters to pass common data between TestStand and the code module. Following is a list of the

different elements within these two structures and how the adapter uses them.

tTestData

- `result`—Set by test function to indicate whether the test passed.
- `measurement`—Numeric measurement that the test function returns.
- `inBuffer`—For passing a string parameter to a test function.
- `outBuffer`—Output message to display in the report.
- `modPath`—Directory path of module containing the test function.
- `modFile`—Filename of module containing the test function.
- `hook`—Reserved (no longer used).
- `hookSize`—Reserved (no longer used).
- `mallocFuncPtr`—Contains a function pointer to `malloc`, which a code module must use to allocate memory for any buffer that it assigns to the `inBuffer`, `outBuffer`, and `errorMessage` fields.
- `freeFuncPtr`—Contains a function pointer to `free`, which a code module must use to free any buffers that the `inBuffer`, `outBuffer`, and `errorMessage` fields point to.
- `seqContextDisp`—A dispatch pointer to the sequence context.
- `seqContextCVI`—A CVI ActiveX Automation handle for the sequence context.
- `stringMeasurement`—String value that the test function returns.
- `structVersion`—Structure version number.
- `replaceStringFuncPtr`—Contains a function pointer to a `ReplaceString` function, which a code module can use to change the value of any buffers that the `inBuffer`, `outBuffer`, and `errorMessage` fields point to.

tErrorData

- `errorFlag`—The test function must set this to `True` if an error occurs.
- `errorLocation`—Reserved (no longer used).
- `errorCode`—The test function can set this to a non-zero value if an error occurs.
- `errorMessage`—The test function can set this to a descriptive string if an error occurs.

Refer to Chapter 12, *Module Adapters*, in the *TestStand User Manual* for more details on these structures.

16. Update the `GetFrequency` function to prompt an operator to enter values into a frequency numeric control and a report text string control using the `frequency.uir` user interface resource file, as follows:
 - a. Open the `frequency.uir` file from the Project window to view the layout of the controls.
 - b. Close the UIR file after you are done, you will not be making any changes to it.
 - c. Update the source code for the `GetFrequency` function as shown. The other functions in the source file do not need modification. Changed lines appear in bold.

```
void __declspec(dllexport) GetFrequency(tTestData
*testData, tTestError *testError)
{
    int error = 0;
    int panelHandle, panel, control;
    char stringBuffer[512];
    panelHandle = LoadPanelEx (0, "frequency.uir",
        PANEL, __CVIUserHInst);
    if (panelHandle < 0)
    {
        error = panelHandle;
        goto Error;
    }

    DisplayPanel (panelHandle);
    GetUserEvent (1, &panel, &control);
    // Assign values from UIR to return data structure
    GetCtrlVal (panelHandle, PANEL_FREQUENCY,
        &testData->measurement);
    GetCtrlVal (panelHandle,
        PANEL_ADDITIONAL_REPORT, stringBuffer);
    testData->replaceStringFuncPtr(&testData->
        outBuffer, stringBuffer);

Error:
    // FREE RESOURCES
    DiscardPanel(panelHandle);

    // If an error occurred, set the error flag to
    // cause a run-time error in TestStand.
    if (error < 0)
```

```

    {
        testError->errorFlag = TRUE;
        // OPTIONALLY SET THE ERROR CODE AND STRING
        testError->errorCode = error;
        testData->replaceStringFuncPtr(
            &testError->errorMessage, "A run-time error
            occurred.");
    }
    return;
}

```

17. Compile the `Frequency.c` source code by selecting **Build»Compile File** to verify that your changes are correct.
18. Save the source code after you successfully compile.
19. Rebuild the DLL by selecting **Build»Create Dynamic Link Library** in the Project window. If the DLL already exists, overwrite the existing copy.

Note *If LabWindows/CVI returns a file permission error when creating the DLL, return to the sequence editor and select the **Unload All Modules** command from the **File** menu. When you make this selection, TestStand unloads all step code modules, which include DLLs, VIs and any other modules the adapter loads. Return to LabWindows/CVI and rebuild the DLL.*

20. Close the external instance of LabWindows/CVI that contains the `frequency.prj` DLL project.
21. In the TestStand sequence editor, close the Edit C/CVI Module Call dialog box by clicking on the **OK** button.
22. In the Sequence File window, right-click on the Clock Frequency Test and select **Edit Limits** from the context menu.

23. In the Edit Numeric Limit Test dialog box, set the comparison type to less than, LT (<), and the measurement value to 100, as shown in Figure 6-14.

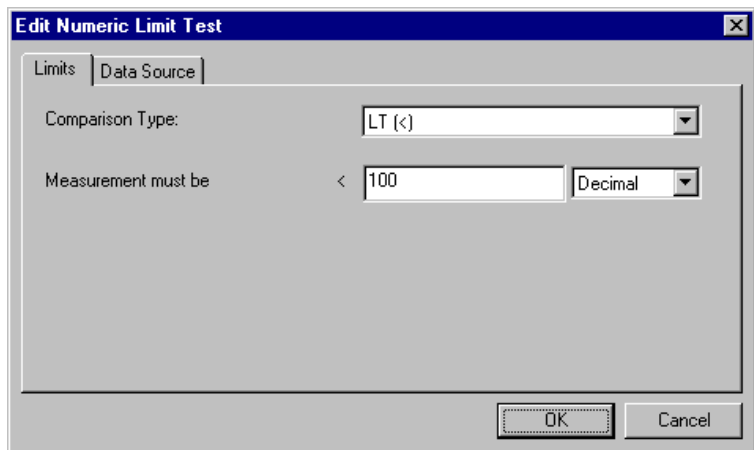


Figure 6-14. Edit Numeric Limits Test Dialog Box

With these settings, TestStand compares the numeric measurement value the code module returns to the constant value of 100. If the comparison is true, the step passes; otherwise, the step fails.

24. Click on **OK** to close the Edit Numeric Limit Test dialog box.
25. Save the sequence by selecting **File»Save As**. Save the sequence as `Sample6.seq` in the `TestStand\Tutorial` directory.
26. Execute the sequence by selecting **Execute»Single Pass**. TestStand launches a new external instance of LabWindows/CVI to execute steps.
27. Click on **Done** in the Test Simulator dialog box.
When TestStand executes the Clock Frequency Test step, the code module displays the user interface panel and waits for input.
28. Type a numeric value of 20 in the Frequency Measurement control.
29. Type any text in the Additional Report Text control.
30. Click on the **Return** button to continue the sequence execution.
31. When the sequence completes the execution, examine the test report. Notice the values for the status, measurement, and report text for the Clock Frequency step.
32. Close the Execution window.

Debugging a CVI Code Module

TestStand not only allows you to debug sequences, but also to step directly into debuggable LabWindows/CVI code modules. In this exercise, you will examine how to debug a LabWindows/CVI code module while executing a sequence in the sequence editor.

1. Set a breakpoint on the Clock Frequency Test step by right-clicking on the step name and selecting **Toggle Breakpoint**. Remember that a breakpoint is enabled for a step if a stop sign icon is visible to the left of the step name in the sequence window.
2. Execute the sequence by selecting **Execute»Single Pass**.
3. Click on **Done** on the Test Simulator prompt. The execution pauses on the Clock Frequency Test step.
4. Click on the **Step Into** toolbar button, which activates an external instance of LabWindows/CVI and enters a breakpoint state on the GetFrequency function, as shown in Figure 6-15.

```

<2> z:\Testexec\Tutorial\frequency.c
File Edit View Build Run Instrument Library Tools Window Options Help

void declspec(dllexport) GetFrequency(tTestData *testData, tTestError *testError)
{
    int error = 0;

    int panelHandle, panel, control;

    char stringBuffer[512];

    panelHandle = LoadPanelEx (0, "frequency.uir", PANEL, __CVIUserHInst);
    if (panelHandle < 0)
    {
        error = panelHandle;
        goto Error;
    }

    DisplayPanel (panelHandle);
    GetUserEvent(1, &panel, &control);

    // Assign values from UIR to return data structure
    GetCtrlVal (panelHandle, PANEL_FREQUENCY, &testData->measurement);
    GetCtrlVal (panelHandle, PANEL_ADDITIONAL_REPORT, stringBuffer);
    testData->replaceStringFuncPtr(&testData->outBuffer, stringBuffer);

Error:
    // FREE RESOURCES
    DiscardPanel(panelHandle);
}

25/100 1C Ins Suspended

```

Figure 6-15. Stepping into the GetFrequency Function

5. Using the **Run»Step Over** command in the menu in LabWindows/CVI, step through the code module.
6. When the user interface resource displays, type a value of 200 in the Frequency Measurement control.
7. Type any text into the Additional Report Text control.
8. Click on the **Return** button.
9. Exit the function by selecting **Run»Finish Function** to return to the sequence execution.
10. After the Clock Frequency Test step executes, TestStand suspends the sequence execution on the Reset Loop Index step. Notice that the status of the Clock Frequency Test step is `Failed` as expected.
11. Select **Debug»Resume** to complete the sequence execution.
12. Close the Execution window and the Sequence File window.
13. If TestStand prompts you to save the sequence file, save the sequence as `Sample6.seq` in the `TestStand\Tutorial` directory.

This concludes this tutorial session. In the next session, you learn how to use the TestStand operator interfaces.

Using Run-Time Operator Interfaces

In this chapter, you learn how to use the LabWindows/CVI operator interface. The features that this chapter discusses also apply to the LabVIEW and the Visual Basic operator interfaces. Refer to Chapter 15, *Run-Time Operator Interfaces* in the *TestStand User Manual* for more information about how to customize a run-time operator interface

TestStand includes three run-time operator interfaces in both source and executable form. Each run-time operator interface is a separate application program. The operator interfaces differ primarily based on the language and ADE in which each is developed. TestStand includes run-time operator interfaces developed in LabVIEW, LabWindows/CVI and Visual Basic. The TestStand run-time operator interfaces are simpler than the sequence editor and are fully customizable.

Loading Sequences

Complete the following steps to load a sequence in a run-time operator interface:

Note *You can use any TestStand operator interface for this tutorial session. The screen shots in this session show the LabWindows/CVI operator interface.*

1. Launch the LabWindows/CVI Operator interface from the Windows taskbar by selecting **Start»Programs»National Instruments TestStand»Operator Interfaces»LabWindows-CVI**.
After the main window for the operator interface displays, a Login dialog box appears.
2. Select the administrator login name, leaving the password empty, and click on the **OK** button.

After you login, the operator interface appears as shown in Figure 7-1.

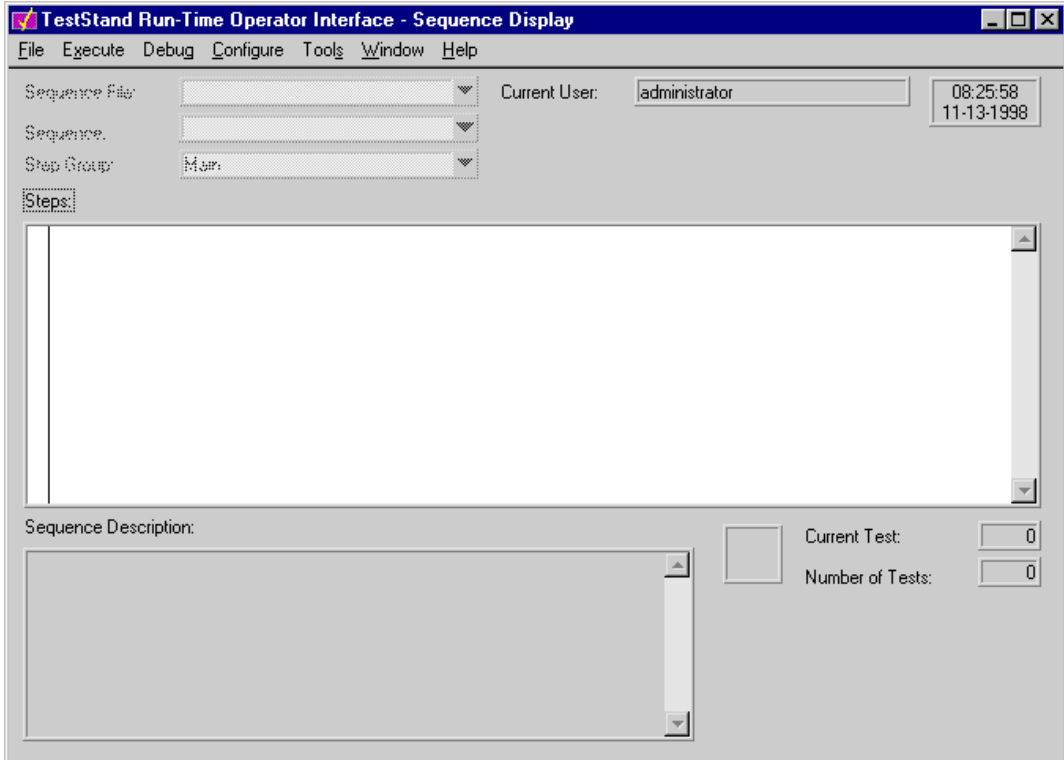


Figure 7-1. LabWindows/CVI Operator Interface

3. Using the mouse, browse through the menu options and notice that the operator interface menus contain many of the commands available from the sequence editor.

Like the sequence editor, the run-time operator interfaces allow you start multiple concurrent executions, set breakpoints, and single-step. Unlike the sequence editor, however, the run-time operator interfaces do not allow you to modify sequences, and they do not display sequence variables, sequence parameters, step properties, and so on.

4. For this exercise, you will not be debugging code modules in LabWindows/CVI. Make sure that the LabWindows/CVI Standard Prototype Adapter is properly configured to execute code modules in the same process as the operator interface, as follows:
 - a. Select **Configure»Adapters**, which displays the Adapter Configuration dialog box.

- b. Select the C/CVI Standard Prototype Adapter in the Configurable Adapters section.
 - c. Click on the **Configure** button, which displays the CVI Standard Adapter Configuration dialog box.
 - d. Enable the Execute Steps In-Process option.
 - e. Click on **OK**, then click on **Done** to close the configuration dialog boxes.
5. Select **File»Open Sequence File**.
 6. Find in the TestStand\Tutorial directory the Sample2.seq sequence file that you created in Chapter 3, *Editing Steps in a Sequence*, and open it.

After you open the sequence file, the operator interface window appears as shown in Figure 7-2.

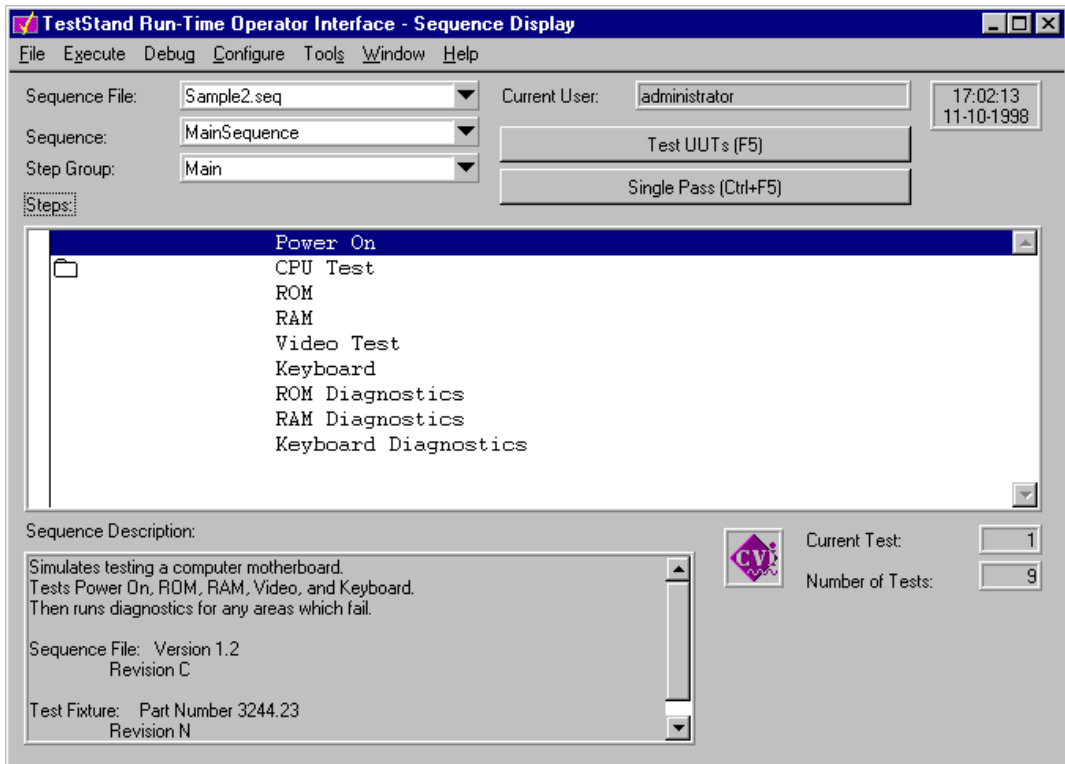


Figure 7-2. Open Sequence in Operator Interface

The Sequence File, Sequence, and Step Group ring controls specify the steps that the operator interface displays in the Steps list box control.

7. Select the `Setup` step group by clicking on the arrow to the right of the right of the Step Group control to view its steps.
8. Reselect `Main` to return to the Main step group.

Running and Debugging Sequences

Complete the following steps to run and debug a sequence in a run-time operator interface:

1. Set a breakpoint on the CPU Test step by clicking on the step and then selecting **Debug»Toggle Breakpoint**. Notice that the letter “B” appears to the left of the step name.
2. Execute the sequence by selecting **Execute»Single Pass**.

When you start the execution, the operator interface displays the execution in a separate window.

- When the execution displays the Test Simulator dialog box, click on **Done**. The execution then pauses at the breakpoint on the CPU Test step, as shown in Figure 7-3.

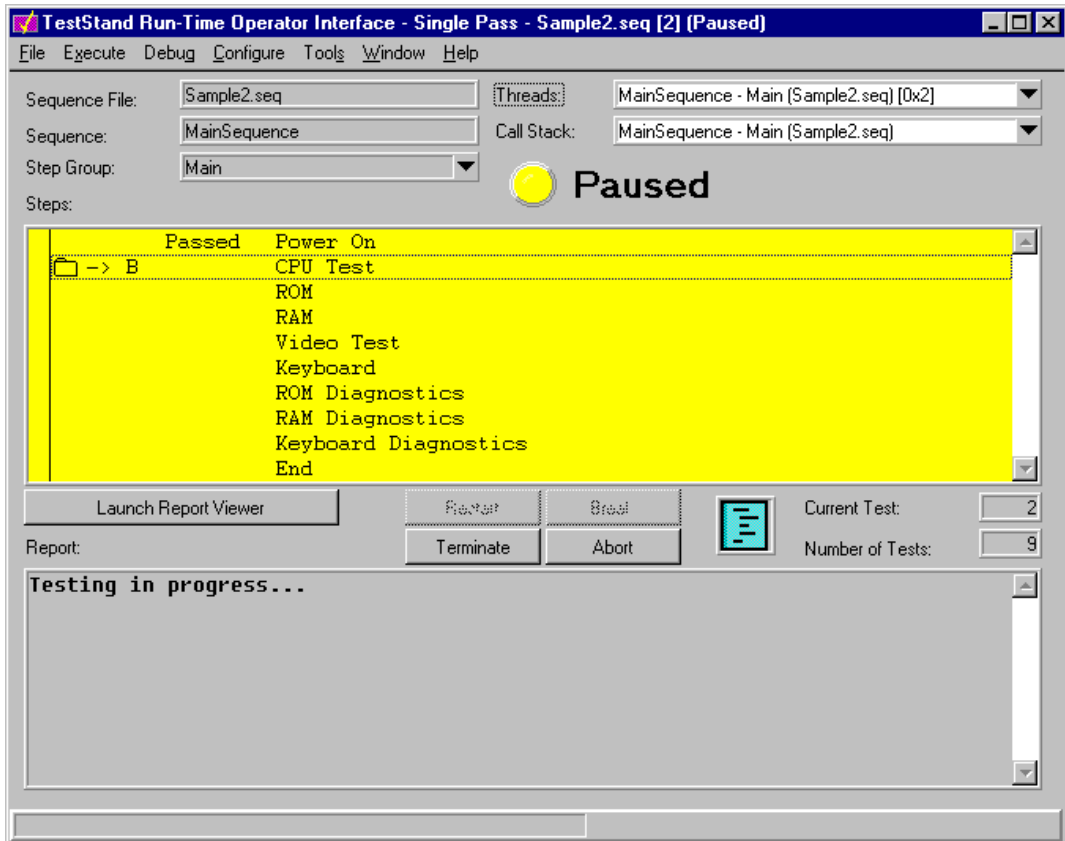


Figure 7-3. Paused Execution in Operator Interface

- Select **Debug»Step Into**. Notice that the Execution window changes and displays the steps in the MainSequence sequence of the SubSequence1.seq sequence file.
 - Single-step a few times by selecting **Debug»Step Over**.
 - Select **Debug»Resume** to complete the execution.
- All of the single-stepping and stepping into source code modules are available to you from the operator interface applications.
- Close the Execution window by selecting **File»Close Execution**.

Running Multiple Executions

Complete the following steps to open a new sequence file and start multiple executions:

1. Select **File»Open Sequence File** and open the `LoopForever.seq` sequence file from the `TestStand\Tutorial` directory. This sequence contains a series of empty steps that continuously loop back to the first step.
2. Start an execution by selecting **Execute»Run “MainSequence”**.
After the new Execution window appears, drag the Execution window off to the side so you can see the Sequence Display window.
3. Start a second execution by again selecting **Execute»Run “MainSequence”** in the Sequence Display window.
If you do not see the **Run “MainSequence”** command in the menu, make sure you have selected the correct window by clicking on its tab.
4. Create a total of four executions. Each of the executions you start executes in a unique thread under the main operator interface process.
5. To terminate all the executions you started, select **Debug»Terminate All**. The **Terminate All** command is available from any of the operator interface windows.
6. Close all the Execution windows by selecting **Window»Close All Completed Executions**.
7. Close the main operator interface window by selecting **File»Exit**.

This concludes this tutorial session. In the next session, you learn how to use callbacks.

Using Callbacks

In this chapter, you learn how to customize the execution of a sequence within TestStand using callbacks. *Callbacks* are sequences that are used to handle common tasks such as serial number inquiry or report logging.

All of the default callback sequences that TestStand includes are provided in source form so that you can edit or replace them to customize TestStand for your particular application. In this tutorial session, you will replace one of the default TestStand callbacks with your own.

Setting Up the Example

If you did not directly proceed from Chapter 7, *Using Run-Time Operator Interfaces*, follow these steps to set up the TestStand sequence editor so you can complete this tutorial session:

1. If the sequence editor is not running, launch the sequence editor.
2. Close all windows in the sequence editor.

Overriding a Process Model Callback

The TestStand process model contains sequences that define operations TestStand performs before and after it tests a UUT. If you want to invoke a sequence in the process model, you can run one of the entry point sequences in the model. The default model entry points are `Test UUTs` and `Single Pass`, as discussed in Chapter 2, *Loading and Running Sequences*, of this manual. The process model contains hooks that allow you to customize the behavior of a process model for each main sequence that uses it without forcing you to edit the process model directly. These hooks are in the form of sequences and are called *model callbacks*.

For example, the default TestStand process model defines a `TestReport` callback that generates the test report for each UUT. Normally, the `TestReport` callback in the default process model file is sufficient because it handles many types of test results. The sequence developer can, however, override the default `TestReport` callback by defining a different

`TestReport` callback in a particular client sequence file. To alter the behavior of the process model for all sequences, you can modify the process model or replace it entirely.

Execution entry points in process models use callbacks to invoke the main sequence in the client sequence file. Each client sequence file must define a sequence by the name of `MainSequence`. The process model contains a `MainSequence` callback that is merely a place holder. The `MainSequence` in the client sequence file overrides the `MainSequence` place holder in the model file.

Figure 8-1 shows the callbacks that the default TestStand process model calls and the order in which TestStand executes the callbacks within the Test UUT execution entry point.

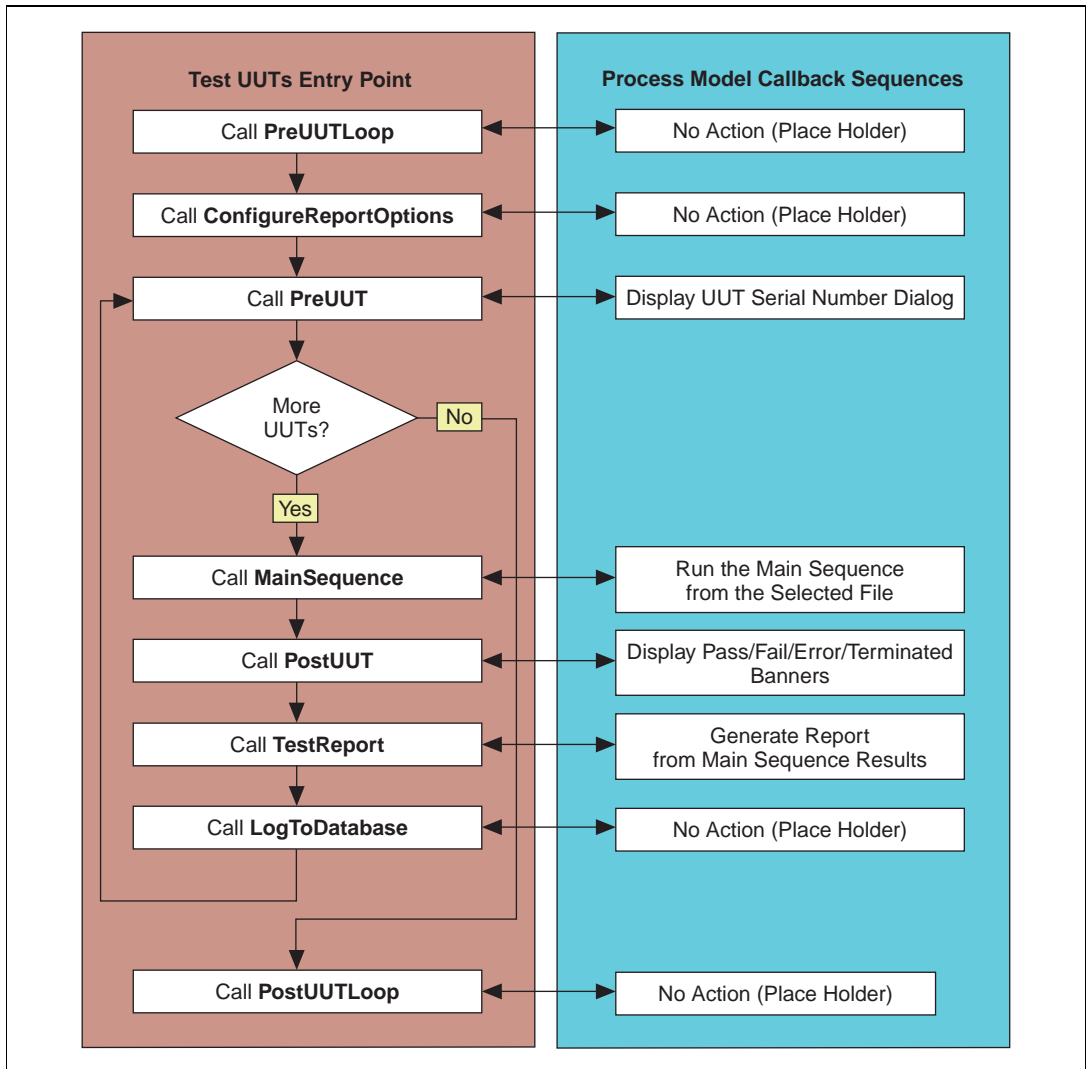


Figure 8-1. TestStand Model Callbacks

Follow these steps to add a PreUUTLoop callback that displays a message popup prompt:

1. Open `TestStandModel.seq` from the `TestStand\Components\NI\Models\TestStandModel` directory. This file is the default process model TestStand uses to execute sequences.
2. Select the sequence `Test UUTs` from the View selector ring. This sequence is the `Test UUTs` entry point that TestStand executes when you select **Execute»Test UUTs**. Notice some of the callback sequences that the `Test UUTs` sequence calls, such as `PreUUTLoop Callback`, `PreUUT Callback`, `MainSequence Callback`, and `PostUUTLoop Callback`, shown in Figure 8-2.

Step	Description	Execution Flow	Comment
Check For Proper Use	Error	Pre, Post	In case someone accidentally tries to...
Clear Report	Action, SetReport(RunState.Execution, "");		Clear the report in case we are rest...
PreUUTLoop Callback	Call PreUUTLoop (<Current File>)		
Get Report Options	Call Get Report Options (<Current File>)		Reads report options, and calls the ...
Get Database Options	Call Get Database Options (<Current File>)	Pre	
Include Limits in Results	Action, AddExtraResult(RunState.Execution...	Pre	Configures the execution to include...
Include Comparison Type in Results	Action, AddExtraResult(RunState.Execution...	Pre	Configures the execution to include...
Next UUT			
Increment UUT Index	Locals.UUT.UUTLoopIndex++		Increment the UUT index. The first...
PreUUT Callback	Call PreUUT (<Current File>)		Display the UUT dialog box.
Goto End of UUT Loop If No More UUTs	Goto 'End of UUT Loop'	Pre, Post	
Determine Report File Path	Action, DetermineReportFilePathName(This...	Pre	Determines the report file path usin...
Put 'Test In Progress' In Report	Action, SetReport(RunState.Execution, Res...		Put the "Test in Progress" string int...
Clear Results List	SetNumElements(Locals.ResultList, 0)		Discard results from previous loop.
Clear Report Local Variable	Locals.Report = ""		Discard the report from the previous...
Get Start Time	Locals.StartTime = Time(), Locals.StartDate ...		
MainSequence Callback	Call MainSequence (<Current File>)		Call MainSequence in the client se...
PostUUT Callback	Call PostUUT (<Current File>)		
Clear Report	Action, SetReport(RunState.Execution, "");		Remove the "Testing in progress" ...
TestReport Callback	Call TestReport (<Current File>)	Pre	Generates the test report string for t...
SetReport	Action, SetReport(RunState.Execution, Loc...		Attaches the report string for the cu...
Write UUT Report	Action, WriteReport(RunState.Report, Loca...	Pre	Writes or appends the report string f...
Log To Database Callback	Call Log To Database (Database.seq)	Pre	
Handle Termination	Testing Terminated for Current UUT	Pre, Post	If you terminate execution in the Ma...
Goto Next UUT	Goto 'Next UUT'	Post	
End of UUT Loop			
PostUUTLoop Callback	Call PostUUTLoop (<Current File>)		
CloseDBHandles	Call Close DB Handles (Database.seq)	Pre	
Read Entire Report	Action, ReadReport(RunState.Report, Loca...	Pre	Read entire report file so that the us...

Figure 8-2. Test UUTs Sequence

3. Right-click on the `PreUUT Callback` step and select **Open Sequence** from the context menu. The `PreUUT Callback` sequence has two steps: `Identify UUT` and `Set Serial Number`.

4. Right-click on the Identify UUT step and select **Run Selected Steps** from the context menu. A familiar-looking dialog box should appear.

This step displays the UUT Information dialog box when you execute a sequence using the Test UUT entry point. If you wanted to change the way in which TestStand obtained a UUT serial number, such as reading it from a bar code, you would replace this callback with your own.

5. Click on **OK** on the UUT Information dialog box.
6. Close the execution window.
7. Select `Test UUTs` again from the View selector ring.
8. Right-click on the PreUUTLoop Callback step and select **Open Sequence** from the context menu. Notice that this callback is empty. The empty sequence is a place holder, so that if you want to add steps that will execute before the UUT loop, you can create them in this callback.
9. Close the `TestStandModel1.seq` sequence file window, and do not save any changes if the sequence editor prompts you to.

You will now override the PreUUTLoop Callback step with your own callback sequence.
10. Open `Sample4.seq` from the `TestStand\Tutorial` directory, which is the sequence file you created and saved in Chapter 5, *Using Variables and Properties*, of this manual.
11. Select **Edit>Sequence File Callbacks** to display the `Sample4.seq` Callbacks dialog box, shown in Figure 8-3.

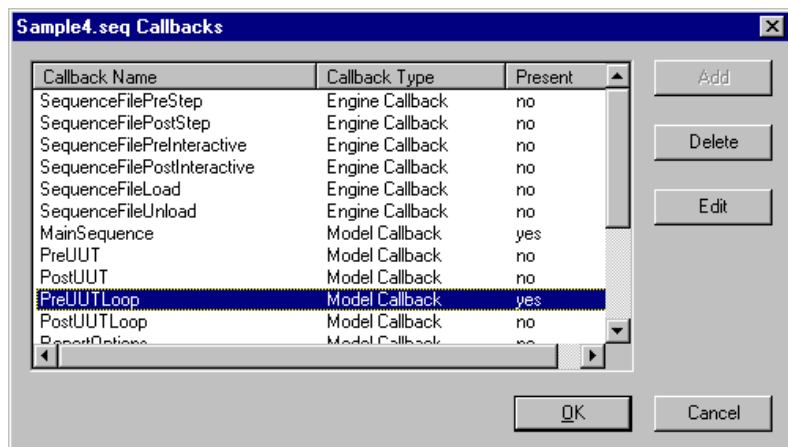


Figure 8-3. Adding Callbacks to a Sequence

12. Select the `PreUUTLoop` callback name.
13. Click on the **Add** button.

Notice that the value in the Present column changes from no to yes. When you click on the **Add** button, the sequence editor creates a new empty callback sequence to your sequence file. Now, when you start an execution using a model entry point, TestStand will call the callback in your sequence file instead of the sequence in the process model.
14. Click on **OK** to close the callbacks dialog box.
15. Select All Sequences in the View selector ring of the sequence file window. Notice that the sequence file now contains two sequences: `MainSequence` and `PreUUTLoop`.
16. Right-click on the `PreUUTLoop` sequence and select **View Contents** from the context menu.
17. Right-click inside the step list pane of the Main tab and select **Insert Step»Message Popup** from the context menu.
18. Rename the new step `Pre UUT Message`.
19. Right-click on the `Pre UUT Message` step and select **Edit Message Settings** from the context menu.
20. In the Title Expression control, enter the literal string "`Pre UUT Loop Callback Message`". You must enclose string literals you enter in any expression field in double quotation marks ("").
21. In the Message Expression, enter the literal string "`Now in the Pre UUT Loop Callback`".
22. Click on **OK** to close the Edit Message Settings dialog box.
23. Save the sequence by selecting **File»Save As**. Save the sequence as `Sample7.seq` in the `TestStand\Tutorial` directory.
24. Execute the sequence by selecting **Execute»Test UUTs**. Notice that the `Pre UUT Loop Callback Message` dialog box is the first prompt TestStand displays.
25. Click on **OK** to close the dialog box. TestStand now displays the UUT Information dialog box from the `PreUUT Callback` sequence in the process model.
26. Enter a serial number and click on **OK**.
27. Run through several iterations of the sequence.

28. Click on **Stop** in the UUT Information dialog box.

Notice that TestStand displays only the Pre UUT Loop Callback Message dialog box once at the very beginning of the execution. The reason is that, as seen in Figure 8-1, the `PreUUTLoop` Callback is executed before the loop, while the `PreUUT` Callback is executed within the loop.

29. Close all windows in the sequence editor.

This concludes this session of the tutorial. In the next session, you learn how to add users and configure user privileges in TestStand.

Adding Users and Setting Privileges

This chapter discusses how to use the TestStand User Manager, and how you can add new users and change their privileges.

Setting Up the Example

If you did not directly proceed from Chapter 8, *Using Callbacks*, close all windows in the sequence editor so you can complete this tutorial session.

Using the User Manager

The TestStand Sequence Editor includes a User Manager for adding and removing users, and for managing the privileges of each user.

Note *The TestStand User Manager is designed to help you implement policies and procedures concerning the use of your test station. It is not a security system and it does not inhibit or control the operating system or third party applications. You must use the system level security features provided by your operating system to secure your test station computer against malicious use.*

In this exercise, you will learn how to view current users and add new ones.

1. Launch the User Manager window by selecting **View»User Manager**. The left pane shows all of the users configured on this station. Expand the `administrator` tree node to display the hierarchy of properties associated with this user. The `Privileges` node contains settings for all actions a user can perform, such as executing sequences, debugging sequences, or adding new users.

When you highlight a node in the tree view, the corresponding property values under the node appear in the list pane on the right. Notice that all the property values for the privileges under the administrator user are `True`.

The privileges are organized in hierarchical groups. A user has a privilege if the property is `True` or if any parent Boolean property is `True`. For example, a user has the privilege to terminate an execution if either the value of the `User.Privileges.Operate.Terminate` property is `True` or the value of the `User.Privileges.Operate` group is `True`.

Figure 9-1 shows the expanded tree view for the properties under the administrator user.

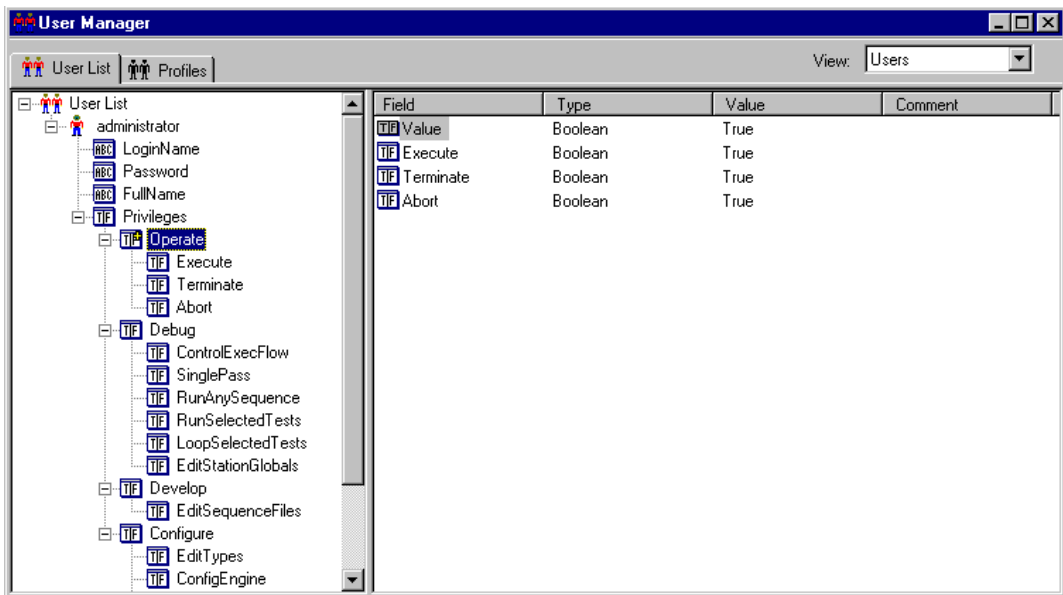


Figure 9-1. User Manager Window

2. Add a new user as follows:
 - a. Click on the User List node in the tree view, which lists the administrator user in the right pane.
 - b. Right-click in the right pane and select **Insert User** from the context menu, which displays the New User dialog box.
 - c. Fill in the Login Name and Full Name controls with your name.
 - d. Type your password into the Password and Confirm Password controls.

Figure 9-2 shows an example of a completed New User dialog box.

Figure 9-2. New User Dialog Box

- e. For the User Profile control, select `Operator`.

User profiles define an initial set of privilege settings to give the new user. By default, the `Operator` profile grants a user the privilege to execute, terminate and abort sequences, but does not grant the privilege to create or debug sequences. TestStand provides four user profiles by default: `Operator`, `Technician`, `Developer`, and `Administrator`.

- f. Click on **OK** to close the New User dialog box.

In addition to adding new users, the User Manager allows you to modify the default profiles and to create new profiles that define a combination of privileges appropriate for your test station.

3. Create a new profile as follows:
 - a. Click on the Profiles tab of the User Manager window. In the list pane you should see the four default profiles.
 - b. Click on the Profiles node in the tree view, which lists the currently defined profiles in the right list pane.
 - c. Right-click on the `Operator` profile in the right pane and select **Copy** from the context menu.
 - d. Right-click in the right pane and select **Paste**.
 - e. Rename the new profile `Senior Operator`. The new profile is identical to the `Operator` profile.

Note *If the Senior Operator profile already exists, the paste operation appends an underscore and a unique number on the end of the name.*

If you make changes to the values in a profile, your changes do not affect the privileges for users that already exist in the user list. After you create a user, you must modify privileges individually. You cannot modify privileges for existing users by changing user profiles.

4. Modify the default privileges for this new profile, as follows:
 - a. Select the new Senior Operator node in the tree view and expand its privilege settings.
 - b. Select the Debug node in the tree view, as shown in Figure 9-3.

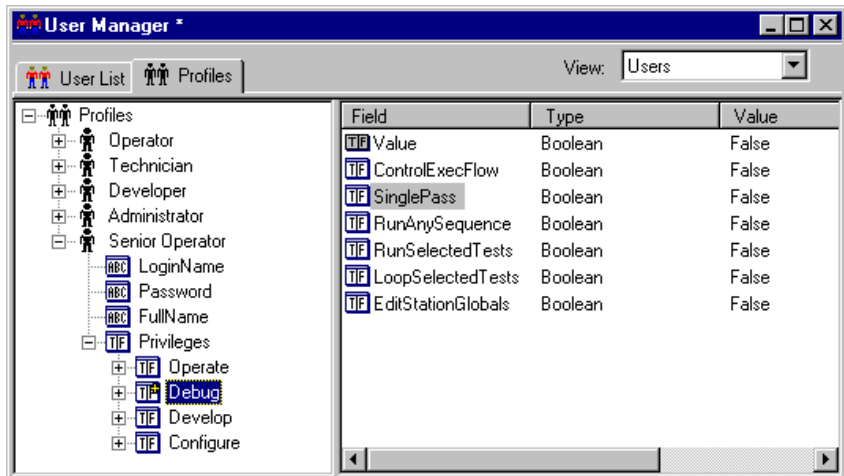


Figure 9-3. Configure Privileges in New Profile

Debug is a Boolean property, which contains additional Boolean subproperties. The value of the Debug property is False and all the subproperties under Debug are also False.

You can set the value of each privilege in the right pane by right-clicking on an item and selecting **Properties** from the context menu.

You can override the privileges in a group and grant a user access to all privileges in a group by setting the value of the parent Boolean property to True.

Set the `SinglePass` property under the `Debug` group to `True` for the `Senior Operator` profile as follows:

- a. Double-click on the `Single Pass` name in the right pane. The `Boolean Properties` dialog box appears.
 - b. Change the value to `True`.
 - c. Click on **OK** to close the dialog box.
5. Add a new user using the `Senior Operator` profile, as follows:
 - a. Click on the `User List` tab.
 - b. Right-click in the right pane and select **Insert User** from the context menu.
 - c. Enter the information in the `New User` dialog box as in the previous step, this time using a different login name.
 - d. Select the `Senior Operator` profile.
 - e. Click on **OK** to close the `New User` dialog box.
 6. Select **File»Login**. The two new users you just added now appear in addition to the `administrator` login name.
 7. Select the user you created with the `Senior Operator` profile.
 8. Enter the appropriate password.
 9. Click on **OK**.
 10. Open `Sample1.seq` from the `TestStand\Tutorial` Directory.
 11. Open the **Execute** menu. Notice that the **Single Pass** menu is selectable, but the command **Run MainSequence** is grayed out because you no longer have the privilege to execute sequences without a model entry point.
 12. Try to right-click in a sequence view to insert a new step, and notice that the insert menu command is also grayed out because your privileges have changed.
 13. Close all of the windows in the sequence editor.
 14. Select **File»Login**.
 15. Log in as the `administrator`. The password for the `Administrator` is an empty string.

This concludes this session of the tutorial. In the next session, you learn how to use the `TestStand ActiveX API` in code modules.

Using ActiveX in Code Modules

This chapter teaches you how to use ActiveX from within a code module in TestStand. If you do not use LabVIEW or LabWindows/CVI, you can skip this chapter and proceed to Chapter 11, *Additional Development Features*.

TestStand gives you various places in which you can store data values. These places are called *variables* and *properties*.

As discussed in Chapter 5, *Using Variables and Properties*, variables are properties that you can freely create in certain contexts. You can have variables that are *global* to a sequence file or *local* to a particular sequence. You can also have *station global* variables. The values of station global variables are persistent across different executions and even across different invocations of the sequence editor or run-time operator interfaces. The TestStand engine maintains the value of station global variables in a file on the run-time computer.

Each step in a sequence can have properties. For example, a step might have an integer error code property. The type of a step determines the set of properties it has. For example, the Numeric Limits Test step contains properties for the comparison type and the high and low limit values.

You can use TestStand variables to share data among tests you write in different programming languages even if they do not have compatible data representations. You can pass values that you store in variables and properties to code modules. Also, you can use the TestStand ActiveX API to access variable and property values directly from code modules. When executing sequences, TestStand maintains a *sequence context* that contains references to all global variables, all 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 sequence context object reference to the code module, you can use the TestStand ActiveX API to access the variables and properties in the sequence context.

In this session, you will learn how create code modules in the LabVIEW and LabWindows/CVI development environments that use the ActiveX API to share data with TestStand.

Using ActiveX in LabVIEW Test Virtual Instruments

In this exercise, you will create a new sequence with two LabVIEW VI steps. The first step generates an array of data and stores it in a TestStand variable, and second step plots the data stored in the variable. This session of the tutorial assumes a general familiarity with the LabVIEW development environment. If you are not using LabVIEW, but you do use LabWindows/CVI, you can skip this section and proceed to the [Using ActiveX in LabWindows/CVI Code Modules](#) section in this chapter.

Note *Make sure you are using the appropriate version of LabVIEW with TestStand. Refer to the `readme.txt` file in the `TestStand\Doc` directory for more details.*

Setting Up the Example

Close all windows in the sequence editor so you can complete this tutorial session.

Creating the Sequence and Virtual Instrument Tests

In this exercise, you will create a new sequence in the sequence editor.

1. Open a new sequence by selecting **File»New Sequence File** in the menu.
2. Save the sequence by selecting **File»Save As**. Save the sequence as `Sample8.seq` in the `TestStand\Tutorial` directory. By saving the sequence file now, you can specify relative paths to code modules instead of absolute paths.
3. Click on the Locals tab of the Sequence File window.

- Right-click in the right pane and insert a numeric array variable, as shown in Figure 10-1.

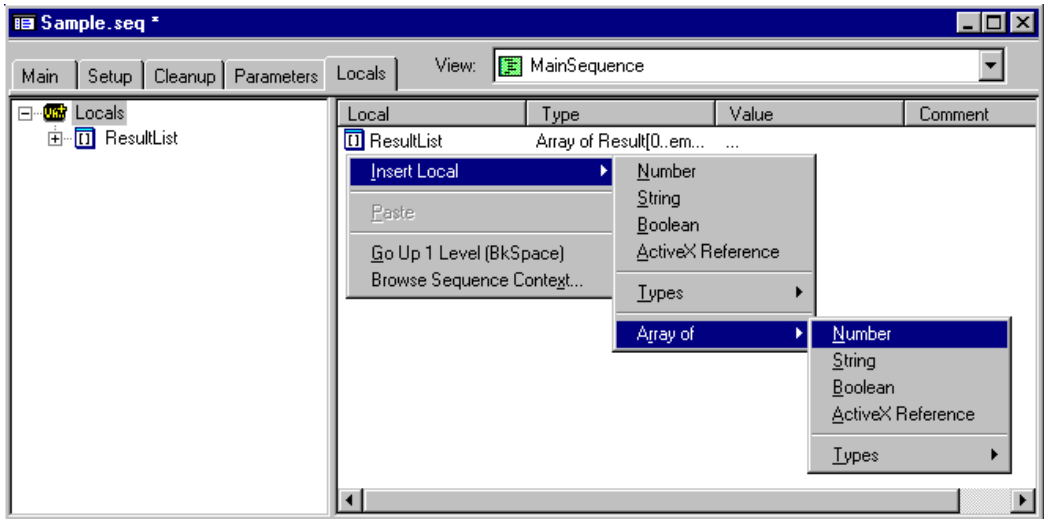


Figure 10-1. Insert Locals Array of Numeric

When you make this selection, the sequence editor displays the Array Bounds dialog box, shown in Figure 10-2.

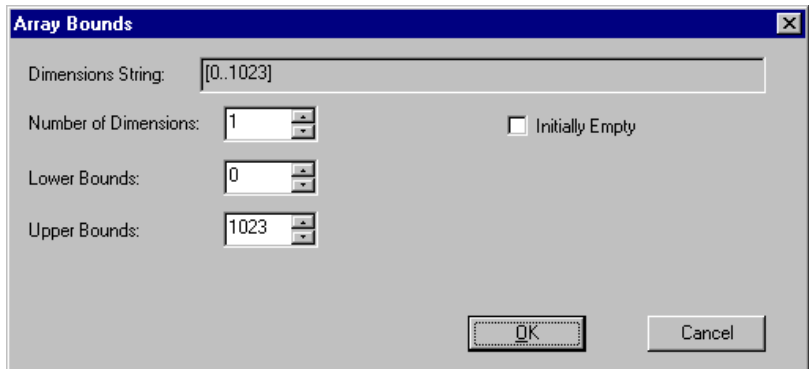


Figure 10-2. Array Bounds Dialog Box

5. Enter a value of 1023 for the Upper Bounds control as shown in Figure 10-2.
6. Click on **OK** to close the dialog box.
7. Right-click on the new variable and select **Rename** from the context menu.
8. Rename the variable `Arraydata`.
9. Click on the Main tab of the Sequence File window. Make sure the LabVIEW Standard Prototype Adapter is selected in the Adapter Selector Ring control.
10. Right-click in the step list and select **Insert Step»Action** to insert a new Action step.
11. Rename the step `Get Array Data From LabVIEW`.
12. Right-click on the `Get Array Data From LabVIEW` step and select **Specify Module** in the context menu.
13. Using the **Browse** button on the Edit LabVIEW VI Call dialog box, select the file `GenerateWaveform.vi` in the `TestStand\Tutorial` directory.
14. On the Edit LabVIEW VI Call dialog box, enable the options `Sequence Context ActiveX Pointer` and `Show VI Front Panel When Called`.
15. Now that you have specified the VI, click on the **Edit Code** button on the Edit LabVIEW VI Call dialog box to open the VI in LabVIEW.

Figure 10-3 shows the front panel of the `GenerateWaveform.vi`. This VI generates a waveform based on the inputs from the front panel.

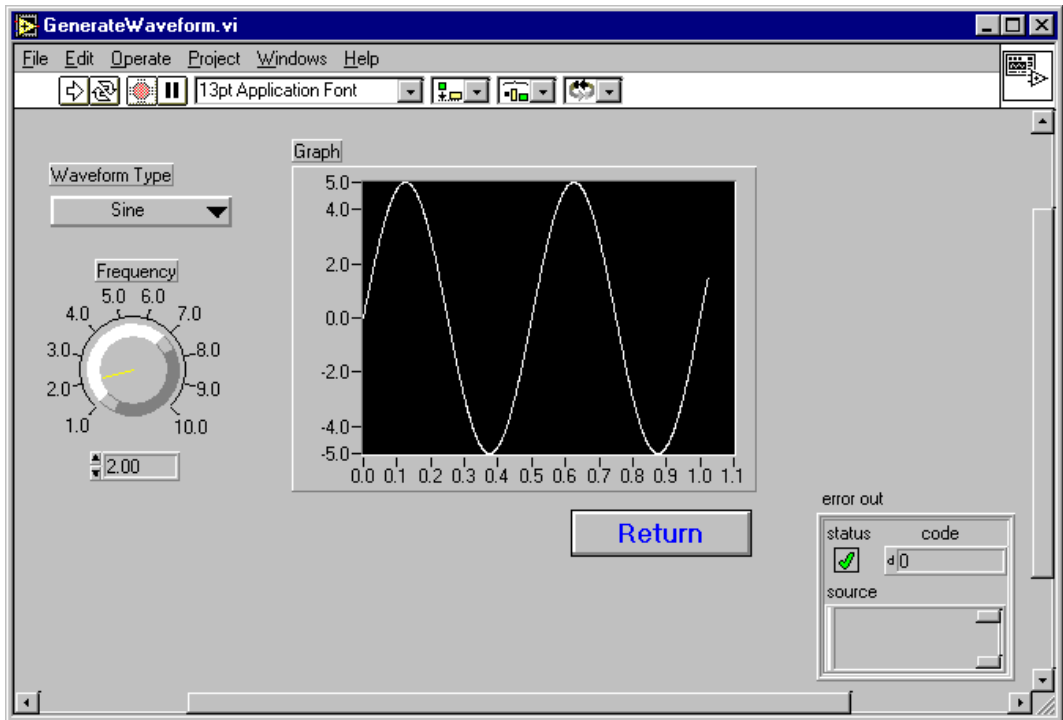


Figure 10-3. `GenerateWaveform.vi` Front Panel

16. Save the VI file as `GenerateTestStandWaveform.vi` in the `TestStand\Tutorial` directory by selecting the **File>Save As** in LabVIEW.

You are saving the file in this manner to preserve the original VI for the next person that uses this tutorial. Later, you will change the specified module in the `TestStand` step to point to this new VI.

When you install `TestStand`, the installation adds a `TestStand.LLB` to the `LabVIEW\User.lib` directory. This VI library contains the `TestStand` LabVIEW Standard Prototype controls, and additional VIs to assist you when calling the `TestStand` ActiveX automation server.

Figure 10-4 shows the TestStand control palette.

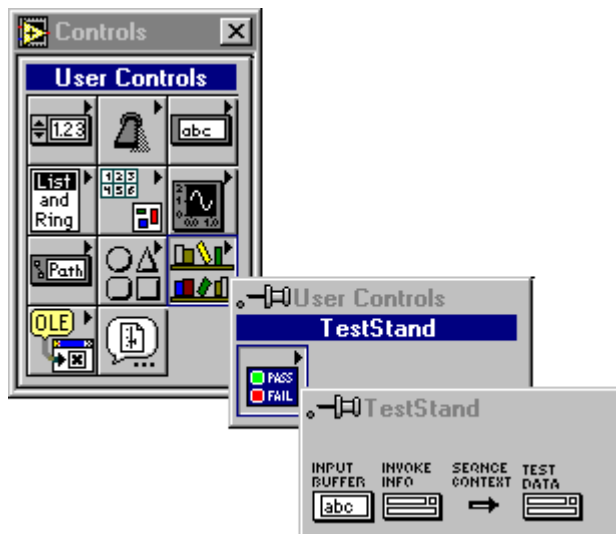


Figure 10-4. TestStand Control Palette

All test VIs that the LabVIEW Standard Prototype Adapter calls must contain Test Data and error out controls. Test VIs also may contain the Sequence Context, Input Buffer, and Invocation Info controls.

17. Add additional controls to the front panel so that the LabVIEW Standard Prototype Adapter can call the VI from TestStand, as follows:
 - a. Drag the Sequence Context and Test Data controls from the TestStand control palette to the VI front panel.

- b. Wire the two new controls to the terminal connector in the upper right corner of the front panel window, as shown in Figure 10-5. If you do not wire the controls to the terminal, TestStand cannot pass data to the controls.

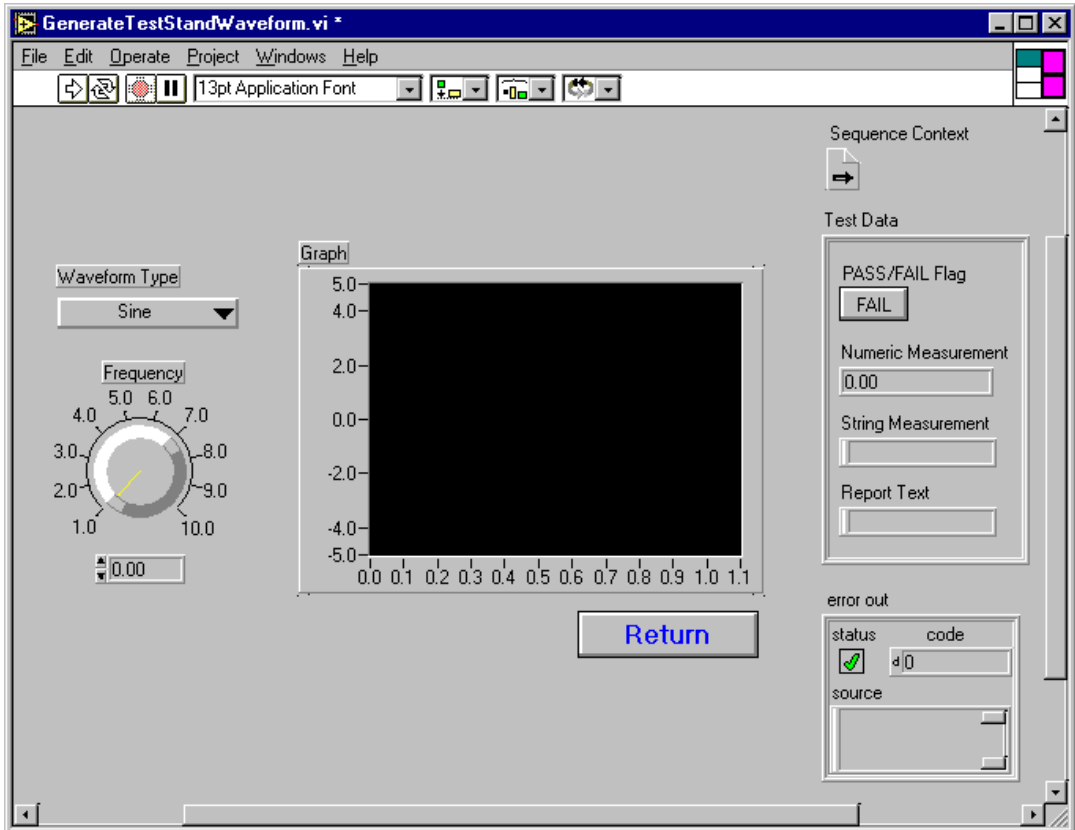


Figure 10-5. GenerateTestStandWaveform.vi Control Panel

- c. The LabVIEW Standard Prototype Adapter does not allow you to pass array data to the Test VI directly through a VI terminal. Instead, the test VI must give the TestStand engine the array data.

18. Use functions in the TestStand function palette to assign the array data to a TestStand variable using the sequence context refnum, as follows:
 - a. Open the block diagram for the test VI as shown in Figure 10-6.

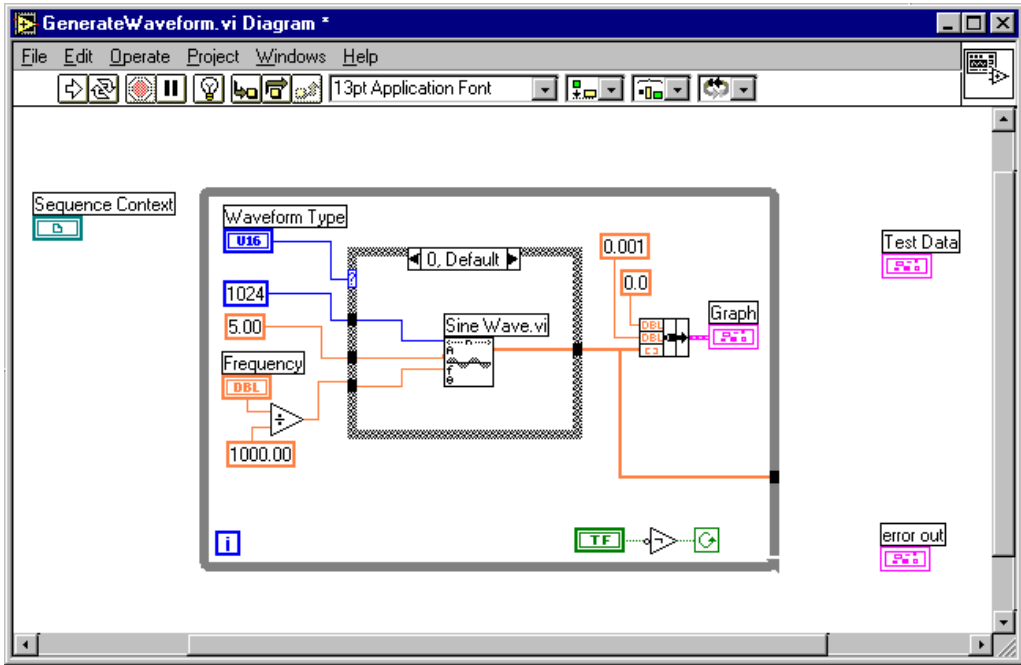


Figure 10-6. GenerateWaveform.vi Block Diagram

You can access the TestStand function palette from the user libraries in LabVIEW as shown in Figure 10-7.

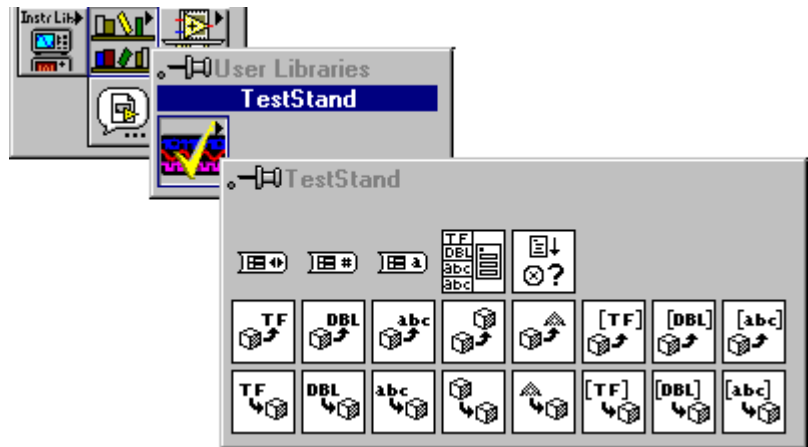


Figure 10-7. TestStand Function Palette

- b. Using the TestStand VIs Set Property Value (Numeric Array).vi and Create Test Data Cluster.vi, update the block diagram as shown in Figure 10-8.

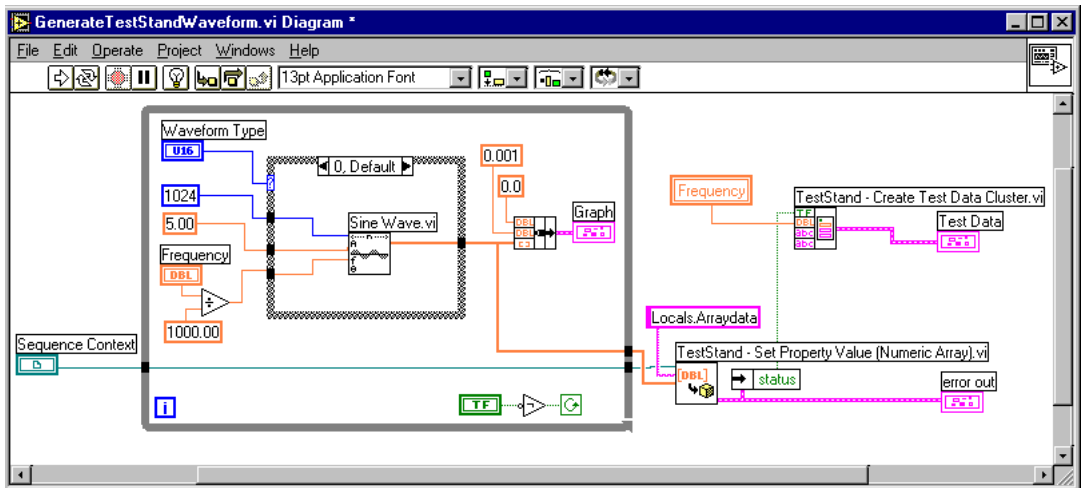


Figure 10-8. GenerateTestStandWaveform.vi Block Diagram

Note *The TestStand function palette contains only wrapper VIs for the most commonly used TestStand ActiveX API methods and properties. Refer to the TestStand ActiveX API online help for more details on the entire ActiveX API.*

- c. Save your changes to the test VI by selecting **File»Save** in LabVIEW. Leave the test VI open in LabVIEW.
19. Return to the Edit LabVIEW VI Call dialog box in the sequence editor.
 20. Re-specify the VI module for the step by clicking on the **Browse** button and selecting the file `GenerateTestStandWaveform.vi` in the `TestStand\Tutorial` directory.
 21. Click on **OK** to close the Edit LabVIEW VI Call dialog box.
 22. Create a second step that plots the data stored in the TestStand variable, as follows:
 - a. Right-click in the step list below the Get Array Data From LabVIEW step and select **Insert Step»Action** to insert a new Action step.
 - b. Rename the new step `Display Array Data` in LabVIEW.
 - c. Right-click on the new step and select the **Specify Module** command in the context menu to display the Edit LabVIEW VI Call dialog box.
 - d. Enter the name `DisplayWaveform.vi` for the VI Pathname control.
 - e. Enable the options `Sequence Context ActiveX Pointer` and `Show VI Front Panel When Called`.
 - f. Click on the **Create Code** button to create and open the test VI in LabVIEW. The sequence editor prompts you for the directory to create the new VI in.
 - g. Select the `TestStand\Tutorial` directory. Notice that the new VI already contains the Sequence Context, Test Data, and error out controls.

- h. Add an XY Graph control and a Dialog button to the VI front panel, as shown in Figure 10-9.

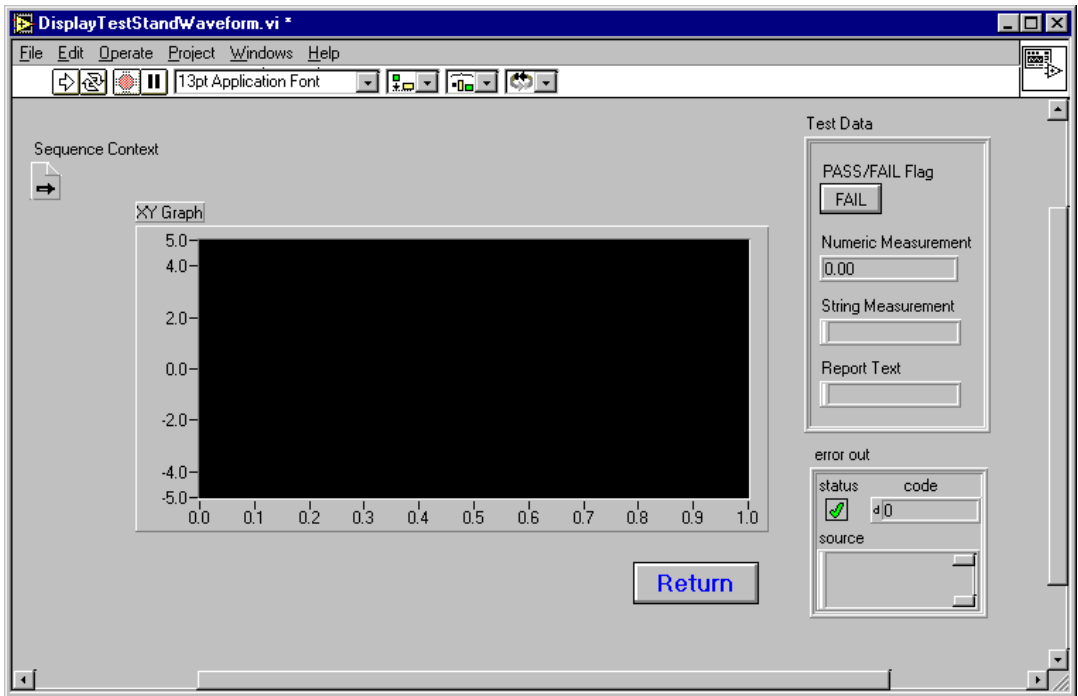


Figure 10-9. DisplayTestStandWaveform.vi Front Panel

- i. Using the TestStand VIs Get Property Value (Numeric Array).vi and Create Test Data Cluster.vi, update the block diagram as shown in Figure 10-10.

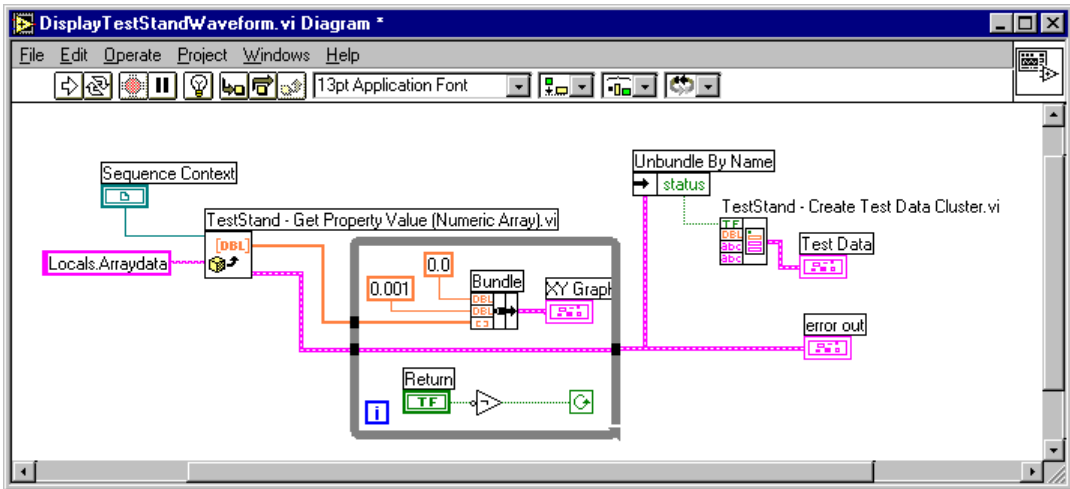


Figure 10-10. DisplayTestStandWaveform.vi Block Diagram

- j. Save your changes to the test VI by selecting **File»Save** in LabVIEW. Leave the test VI open in LabVIEW.
23. Return to the Edit LabVIEW VI Call dialog box in the sequence editor.
24. Click on **OK** to close the dialog box.
25. From the main sequence editor window, select **File»Save** to save the sequence file changes.

Running the Sequence

You will now run the sequence to verify that the code module properly generates and displays the array data.

1. Execute the sequence by selecting **Execute»Run MainSequence**.
2. When the front panel for the two VI tests appears, click on the **Return** button to continue. Notice that the second VI front panel displays the waveform you specified on the first front panel.
3. Close the Execution window after the execution completes.

4. Rerun the sequence and view the data stored in the TestStand array variable as follows:
 - a. Place a breakpoint on the Display Array Data in LabVIEW step by right-clicking on the step in the Sequence File window and selecting **Toggle Breakpoint** from the context menu.
 - b. Execute the sequence again by selecting **Execute»Run MainSequence**.
 - c. When the front panel for the first VI test appears, click on the **Return** button to continue.
 - d. When the execution stops at the breakpoint, click on the Context tab and select the `Locals.Arraydata` variable, as shown in Figure 10-11. Notice the non-zero values for the contents of the array.

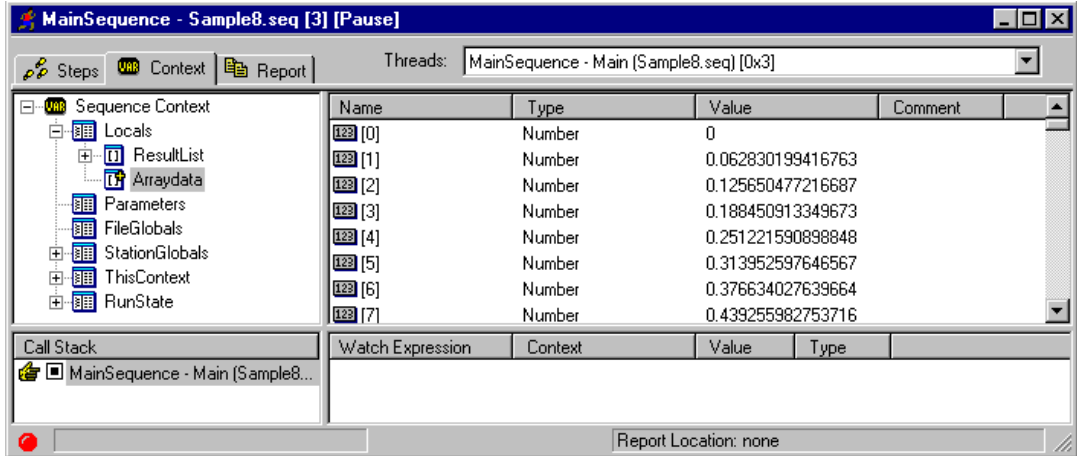


Figure 10-11. Locals.Arraydata

5. Select **Debug»Resume** to continue.
6. When the front panel for the second VI test appears, click on the **Return** button to finish the execution.
7. Close the Execution window after the execution completes.
8. Close all windows in the sequence editor.

If you develop code modules in LabWindows/CVI, proceed to the next section in this chapter and complete the *Setting Up the Example* section. Otherwise, go directly to Chapter 11, *Additional Development Features*, in this manual.

Using ActiveX in LabWindows/CVI Code Modules

You will now create a new sequence with two LabWindows/CVI steps that generate an array of data, and plot the data. This session of the tutorial assumes a general familiarity with the LabWindows/CVI development environment.

Note *Make sure that you are using the appropriate version of LabWindows/CVI with TestStand. Refer to the `readme.txt` file in the `TestStand\Doc` directory for more details.*

Setting Up the Example

Close all windows in the sequence editor so you can complete this tutorial session.

Creating the Sequence and Tests

You will now create a new sequence in the sequence editor.

1. Open a new sequence by selecting **File»New Sequence File**.
2. Save the sequence by selecting **File»Save As**. Save the sequence as `Sample9.seq` in the `TestStand\Tutorial` directory. By saving the sequence file now, you can specify relative paths to code modules instead of absolute paths.
3. Click on the Locals tab of the Sequence File window.

- Right-click in the right pane and insert a numeric array variable, as shown in Figure 10-12.

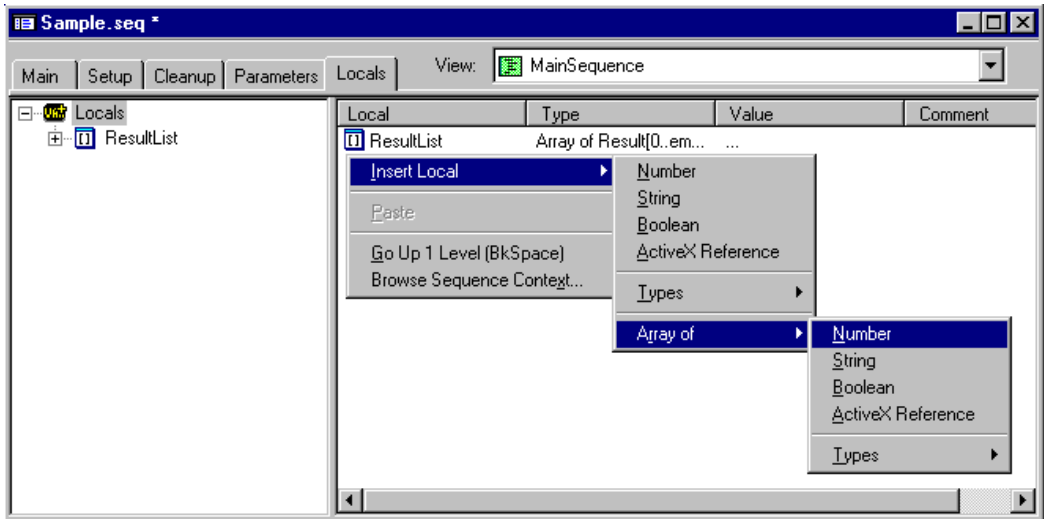


Figure 10-12. Insert Locals Array of Numeric

When you make this selection, the sequence editor displays the Array Bounds dialog box.

- Enter a value of 1023 for the Upper Bounds control, as shown in Figure 10-13.

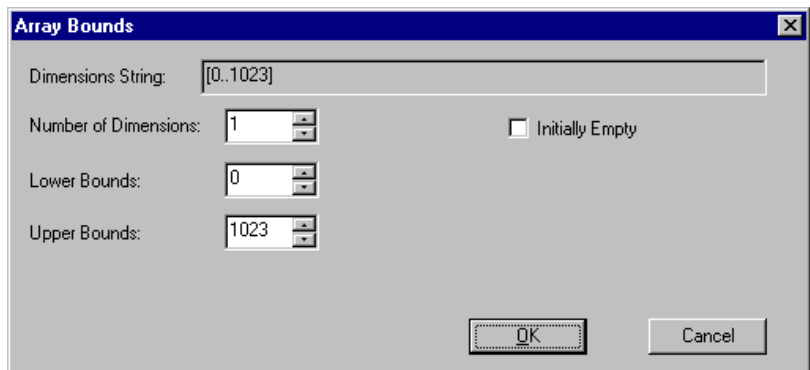


Figure 10-13. Array Bounds Dialog Box

6. Click on **OK** to close the dialog box.
7. Right-click on the variable and select **Rename** from the context menu.
8. Rename the variable `Arraydata`.
9. Click on the Main tab of the Sequence File window. Make sure the C/CVI Standard Prototype Adapter is selected in the Adapter Selector Ring control.
10. Right-click in the step list and select **Insert Step»Action** to insert a new Action step.
11. Rename the step `Get Array Data From CVI`.
12. Right-click on the new step and select the **Specify Module** command in the context menu.
13. On the Module tab in the Module Type ring control, select Dynamic Link Library (*.dll).
14. For the Module Pathname control, enter the name `UsingActiveXInCVI.DLL`.
15. For the Function Name control, enter the name `GenerateTestStandWaveform`.
16. Enable the Pass Sequence Context option.
17. On the Source Code tab, click on the **Create Code** button. When the adapter prompts you for a pathname for the project, enter `UsingActiveXInCVI.prj` in the `TestStand\Tutorial` directory.
18. When the dialog box prompts you for a pathname for the source file, enter the name `UsingActiveXInCVI.c`.
After you enter the name, TestStand does the following:
 1. Launches an external instance of LabWindows/CVI
 2. Creates a new project file in LabWindows/CVI
 3. Creates the source file
 4. Adds the source file and the TestStand support instrument drivers to the project
 5. Generates a template `GenerateTestStandWaveform` function in the source file

Figure 10-14 shows the generated function in the source file.

```

<1> z:\TestStand\Tutorial\UsingActiveX\InCVI.c
File Edit View Build Run Instrument Library Tools Window Options Help

#include "stdtst.h"
#include "tsutil.h"

void __declspec(dllexport) __stdcall GenerateTestStandWaveform(tTestData *testData, tTestError
{
    int error = 0;
    // ErrMsg errMsg = {'\0'};
    // ERRORINFO errorInfo;

    // INSERT YOUR SPECIFIC TEST CODE HERE
    // char *lastUserName = NULL;

    // The following code shows how to accesses a property or variable via the TestStand Active
    // tsErrChk (TS_PropertyGetValString(testData->seqContextCVI, &errorInfo,
    //                                     "StationGlobals.TS.LastUserName",
    //                                     0, &lastUserName));

Error:
    // FREE RESOURCES
    // if (lastUserName != NULL)
    //     CA_FreeMemory(lastUserName);

    // If an error occurred, set the error flag to cause a run-time error in TestStand.
    if (error < 0)
    {
        testError->errorFlag = TRUE;

        // OPTIONALLY SET THE ERROR CODE AND STRING
        // testError->errorCode = error;
        // testData->replaceStringFuncPtr(&testError->errorMessage, errMsg);
    }

    return;
}

4/35 38 CS Ins 4

```

Figure 10-14. Generated GenerateTestStandWaveform Source

Note *If someone else previously completed this session of the tutorial, the project and source files may already exist. Replace any existing copies of the files. When LabWindows/CVI prompts you to replace the existing function, click on the Replace button to continue with this tutorial session.*

- Update the GenerateTestStandWaveform function to prompt an operator to enter the number of sine cycles to initialize the array with. Update the source code for the function as follows. Changed lines appear in bold.

```

void __declspec(dllexport) __stdcall
    GenerateTestStandWaveform(tTestData *testData,
        tTestError *testError)
{
    int error = 0;
    ErrMsg errMsg = {"Error occurred generating
        waveform"};

```

```

ERRORINFO errorInfo;

int i;
double cycles = 2.0;
char buffer[32];
double Arraydata[1024];
VARIANT variantData;

// Prompt for number of cycles
PromptPopup ("Frequency", "Please input number of
cycles for the array?", buffer, 32);
sscanf(buffer, "%lf", &cycles);

// Initialize C array
for (i=0; i<1024; i++)
    Arraydata[i] = sin((2*3.14) * i * cycles /
    1024);

// Copy C array to VARIANT
CA_VariantSet1DArray (&variantData, CAVT_DOUBLE,
    1024, Arraydata);

// The following code shows how to accesses a
// property or variable via the TestStand
// ActiveX API
tsErrChk (TS_PropertySetValVariant
    (testData->seqContextCVI, &errorInfo,
    "Locals.Arraydata", 0, variantData));

Error:
    // FREE RESOURCES

    // If an error occurred, set the error flag to
    // cause a run-time error in TestStand.
    if (error < 0)
    {
        testError->errorFlag = TRUE;

        // OPTIONALLY SET THE ERROR CODE AND STRING
        testError->errorCode = error;
        testData->replaceStringFuncPtr(&testError->
        errorMessage, errMsg);
    }

    return;
}

```

20. Compile the source code by selecting **Build>Compile File** to verify that your changes are correct.
21. Save the source code after you successfully compile.

22. Add another function that will get the array data from the TestStand engine and plot the data. Type the following additional function into the source file.

```
void __declspec(dllexport) __stdcall
DisplayTestStandWaveform(CAObjHandle
seqContextCVI)
{
    int error = 0;
    ErrMsg errMsg = {'\0'};
    ERRORINFO errorInfo;

    int elements = 0;
    double *Arraydata = NULL;
    VARIANT variantData;

    // The following code shows how to accesses a
    // property or variable via the TestStand
    // ActiveX API
    tsErrChk (TS_PropertyGetValVariant
        (seqContextCVI, &errorInfo, "Locals.Arraydata",
        0, &variantData));

    // Copy C array to VARIANT
    CA_VariantGet1DArray (&variantData, CAVT_DOUBLE,
        &Arraydata, &elements);

    WaveformGraphPopup ("Waveform From TestStand",
        Arraydata, elements, VAL_DOUBLE, 1.0, 0.0,
        0.0, 1.0);

Error:
    if (Arraydata)
        CA_FreeMemory(Arraydata);

    return;
}
```

23. Compile the source code by selecting **Build»Compile File** to verify that your changes are correct.
24. Save the source code after you successfully compile.
25. To build the DLL, select **Build»Create Dynamic Link Library** in the Project window.
26. Click on the **OK** button in the Create Dynamic Link Library dialog box.
27. After the DLL build is complete, return to the sequence editor.

28. Click on **OK** to close the Edit C/CVI Module Call dialog box to return to the Main step group view.

Note *If LabWindows/CVI returns a file permission error when creating the DLL, return to the sequence editor and select the **Unload All Modules** command from the **File** menu. When you make this selection, TestStand unloads all step code modules, which includes DLLs, VIs and any other modules the adapter loads. Return to LabWindows/CVI and rebuild the DLL.*

29. Select the DLL Flexible Prototype Adapter in the Adapter Selector Ring control.
30. Right-click in the step list below the Get Array Data From CVI step and select **Insert Step»Action** to insert a new Action step.
31. Rename the step `Display Data In CVI`.
32. Right-click on the new step and select the **Specify Module** command in the context menu.
33. On the Module tab, click on the **Browse** button next to the DLL Pathname control.
34. Select the `UsingActiveXInCVI.dll` that you created.
35. Select the `DisplayTestStandWaveform` function in the Function Name ring control. TestStand displays a message saying the function does not have parameter information in the DLL.
36. Click on the **New** button to create a parameter for the function.
37. Rename the `arg1` parameter `seqContextCVI`.
38. Select **Object** in the Category ring control.
39. Select **CVI ActiveX Automation Handle** in the Object Type ring control.
40. Enter the expression, `ThisContext`, in the Value control.

Note *The purpose of this session is to demonstrate how to use the sequence context within a code module. When you use the DLL Flexible Prototype Adapter, you can pass the array directly to the DLL as a parameter to the function.*

41. Click on **OK** to close the Edit DLL Call dialog box.
42. In the main sequence editor window, select **File»Save** to save the sequence file changes.

Running the Sequence

You will now run the sequence to verify that the code module properly generates and displays the array data.

1. Execute the sequence by selecting **Execute»Run MainSequence**.
2. Enter a value of 5 for the number of sine cycles to generate for the array data.
3. Click on the **OK** button to continue.
4. After the second step displays the graph, click on **OK** to continue.
5. Close the Execution window after the execution completes.
6. Rerun the sequence execution and view the data stored in the array variable in TestStand as follows:
 - a. Place a breakpoint on the `Display Data In CVI` step by right-clicking on the step in the Sequence File window and selecting **Toggle Breakpoint** from the context menu.
 - b. Execute the sequence again by selecting **Execute»Run MainSequence**.
 - c. Enter a value of 5 for the number cycles to generate in the array.
 - d. Click on the **OK** button to continue.
 - e. When the execution stops at the breakpoint, click on the Context tab.

- f. Select the `Locals.Arraydata` variable as shown in Figure 10-15. Notice the non-zero values for the contents of the array.

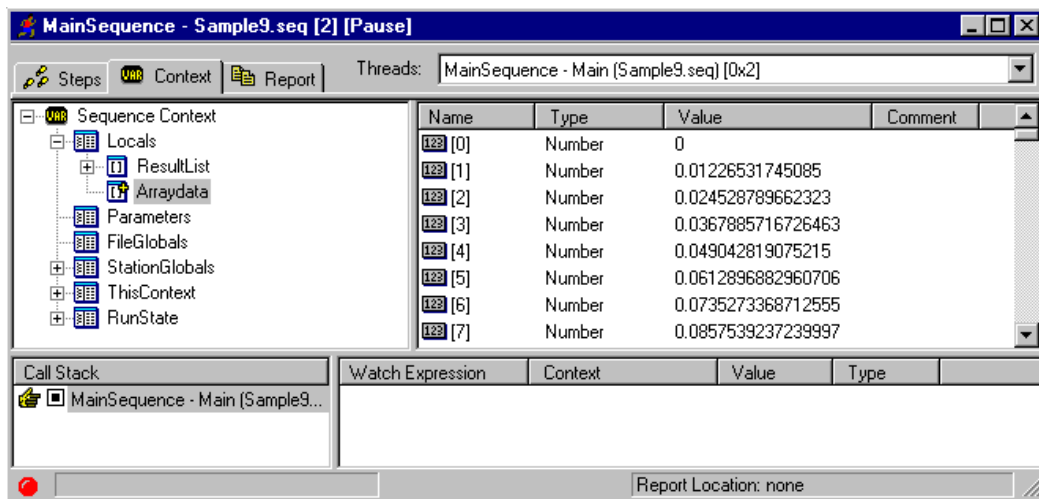


Figure 10-15. `Locals.Arraydata` Values

7. Select **Debug»Resume**.
8. After the graph window appears, click on **OK** to continue.
9. Close the Execution window after the execution completes.
10. Close all windows in the sequence editor.

This concludes this tutorial session. In the next session, you learn some more advanced features of TestStand to use when developing and debugging sequences.

Additional Development Features

In this chapter, you learn how to use some of the more advanced features available when developing and debugging sequences. Also, you learn how to interactively execute steps, and how to dynamically call a sequence by name.

Setting Up the Example

Close all windows in the sequence editor so you can complete this tutorial session.

Interactive Execution

In this exercise, you will run selected steps from a Sequence File window and interactively execute steps while paused at a breakpoint during an execution.

Running Selected Steps as a Separate Execution

In this exercise, you will execute selected steps in a Sequence File window.

1. Open `Sample1.seq` from the `TestStand\Tutorial` directory.
2. After you open the sequence file, place a breakpoint on the Power On step by clicking to the left of the step icon or by right-clicking on the step and selecting **Toggle Breakpoint** from the context menu.
3. Select the Power On, ROM, and ROM Diagnostics steps by holding down the <Ctrl> key and left-clicking on each step.

- After you make these selections, the Sequence File window appears, as shown in Figure 11-1.

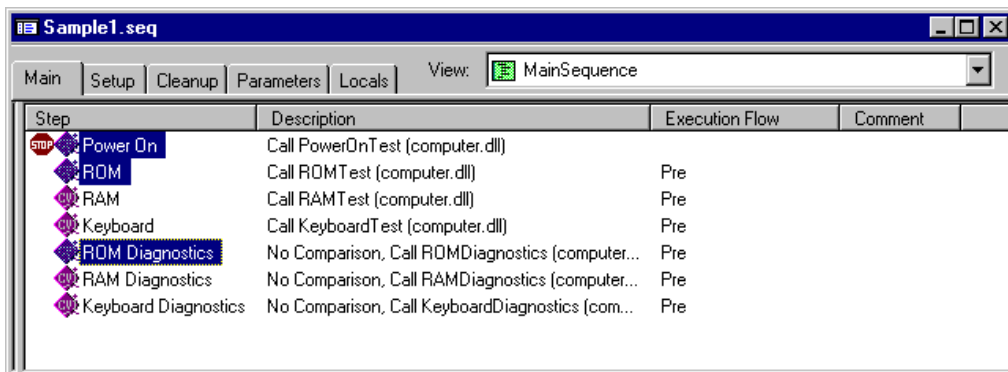


Figure 11-1. Selecting Multiple Steps in a Sequence File Window

- Select **Execute»Run Selected Steps**. When you make this selection, TestStand starts a new execution.
When you run selected steps from a Sequence File window, by default TestStand executes the Setup and Cleanup step groups in addition to the steps in the Main step group.
- When the Test Simulator dialog box appears, click on **Done** to close the dialog box.

- The execution now enters the breakpoint on the Power On step, as shown in Figure 11-2.

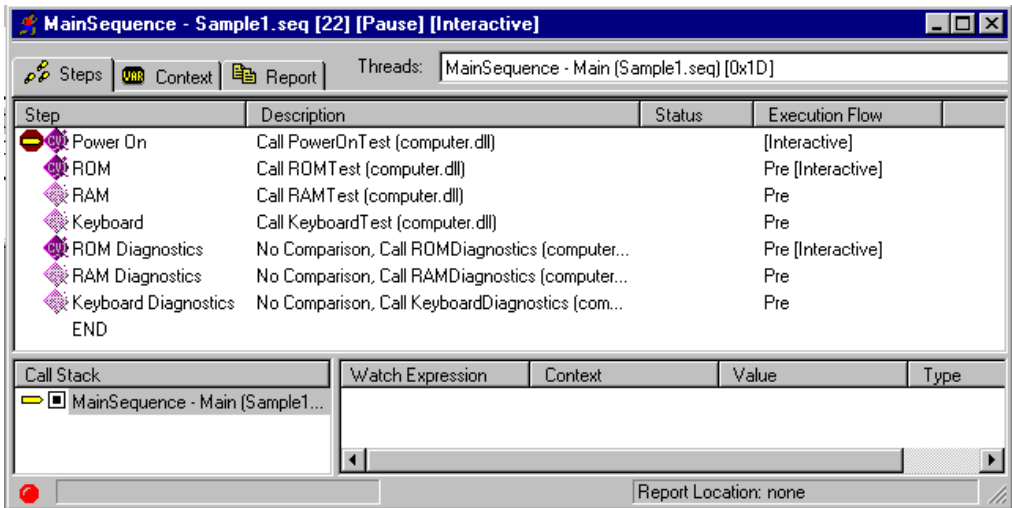


Figure 11-2. Breakpoint During Interactive Execution

Notice that the pointer for the interactive execution is a narrow arrow instead of the yellow arrow that the sequence editor uses for a normal execution.

- Single-step the execution twice by selecting **Debug»Step Over**. Notice that the execution executes only the steps that you previously selected and that the non-selected step icons are dimmed.
- Complete the execution by selecting **Debug»Resume**. Notice that TestStand ignores the preconditions for the ROM Diagnostics step and runs the step even though the ROM step passed.
- Close the Execution window after the execution completes.

Running Selected Steps During an Execution

In this exercise, you will interactively execute selected steps while paused at a breakpoint during an execution.

- Select **Execute»Single Pass** to start a new execution.
- When the Test Simulator dialog box appears, select the ROM test to fail.
- Click on **Done** to close the dialog box. The execution now enters the breakpoint on the Power On step.

4. Using **Debug»Step Over**, continue the execution until you reach the RAM Diagnostics step. Notice that the ROM step failed.
5. Place a second breakpoint on the ROM step in the Execution window by clicking to the left of the step icon or by right-clicking on the step and selecting **Toggle Breakpoint** from the context menu.
6. Now select the ROM, and ROM Diagnostics steps by holding down the <Ctrl> key and left-clicking on each step.
7. Right-click on the ROM Diagnostics step and select **Loop on Selected Steps** from the context menu, as shown in Figure 11-3.

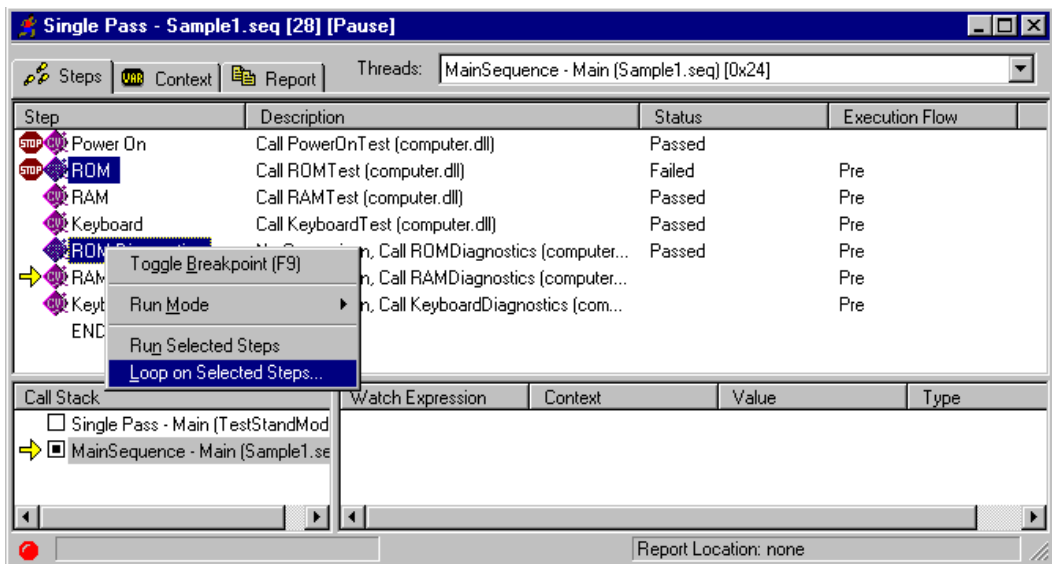


Figure 11-3. Loop on Selected Steps During Execution

8. In the Loop on Selected Steps dialog box, enter the number 100 in the Loop Count control.
9. Click on **OK** to close the dialog box. The sequence editor starts an interactive execution for the selected steps and enters a paused state on the breakpoint for the ROM step. Notice that the yellow arrow icon is still on the RAM Diagnostics step and a new narrow arrow icon is now pointing to the ROM step.
10. Single-step the interactive execution by selecting **Debug»Step Over**. Notice that the interactive execution toggles between only the ROM and the ROM Diagnostics steps.

11. Because the status of the ROM step does not pass and continues to fail, select **Debug»Terminate Interactive Execution**. When you make this selection, the TestStand returns the execution to a paused state on the RAM Diagnostics step.
12. Force the execution to continue from a step other than the currently paused step as follows:
 - a. Left-click on the ROM step so that it is the only highlighted step.
 - a. Right-click on the ROM step and select **Set Next Step** from the context menu. Notice that the yellow arrow icon moves from the RAM Diagnostics step to the ROM step.
 - b. Select **Debug»Step Over** to single-step once and notice that the execution executes the ROM step instead of the RAM Diagnostics step.
13. Complete the execution by selecting **Debug»Resume**.
When the report file completes, notice that the report contains entries for each interactively executed step.
14. Close all windows in the sequence editor and do not save any changes to the sequence file.

Calling Sequences Dynamically and Passing Parameters

In the following exercise, you will add a step to a sequence that dynamically runs one of two sequences.

Adding a Step to Sequence

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

1. Open `Sample1.seq` from the `TestStand\Tutorial` directory.
2. Click on the **Locals** tab of the sequence window.
3. Right-click in the right pane and select **Insert Local»String** from the context menu.
4. Rename the local variable `CPUType`.
5. Click on the Main tab in the Sequence File window to display the steps in the Main step group.

6. Right-click on the Power On step and select **Insert Step»Message Popup** from the context menu.
7. Rename the new step `Select CPU Type`.
8. Right-click on the new step and select **Edit Message Settings** from the context menu.
9. Change the following control values on the Configure Message Box 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

10. Click on **OK** to close the Configure Message Box Step dialog box.
11. Right-click on the Select CPU Type step and select **Properties** from the context menu.
12. Click on the Expressions tab.
13. Enter the following expression in the Post Expression control.

```
Locals.CPUType = ((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 on.

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

Title Expression	"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

19. Click on **OK** to close the Configure Message Box Step dialog box.
20. Right-click on the Specify Number of CPUs step and select the **Insert Step»Sequence Call** command in the context menu.
21. Rename the step CPU Test.
22. Right-click on the new CPU Test step and select **Specify Module** from the context menu.
23. Enable the Specify Expressions for Pathname and Sequence option.
24. Enter the following values for the File Pathname Expression and Sequence Expression controls:

File Pathname Expression `Locals.CPUType + "Processor.seq"`

Sequence Expression `"MainSequence"`
25. Select the prototype for the sequence call by clicking on the **Load Prototype** button.
26. Click on the **Browse** button in the Load Sequence Prototype dialog box.
27. Select the `AMDProcessor.seq` sequence file.
28. Click on **OK** twice to close both the Select Sequence File and the Load Sequence Prototype dialog boxes.

Notice that TestStand populates the Parameters section with the parameter list for the sequence.
29. Click on the `CPUsInstalled` parameter.
30. Select the Enter Expression option.
31. Enter the following expression into its string control, or click on **Browse** and find the property in the Expression Browser dialog box:


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

Figure 11-4 shows the completed Edit Sequence Call dialog box.

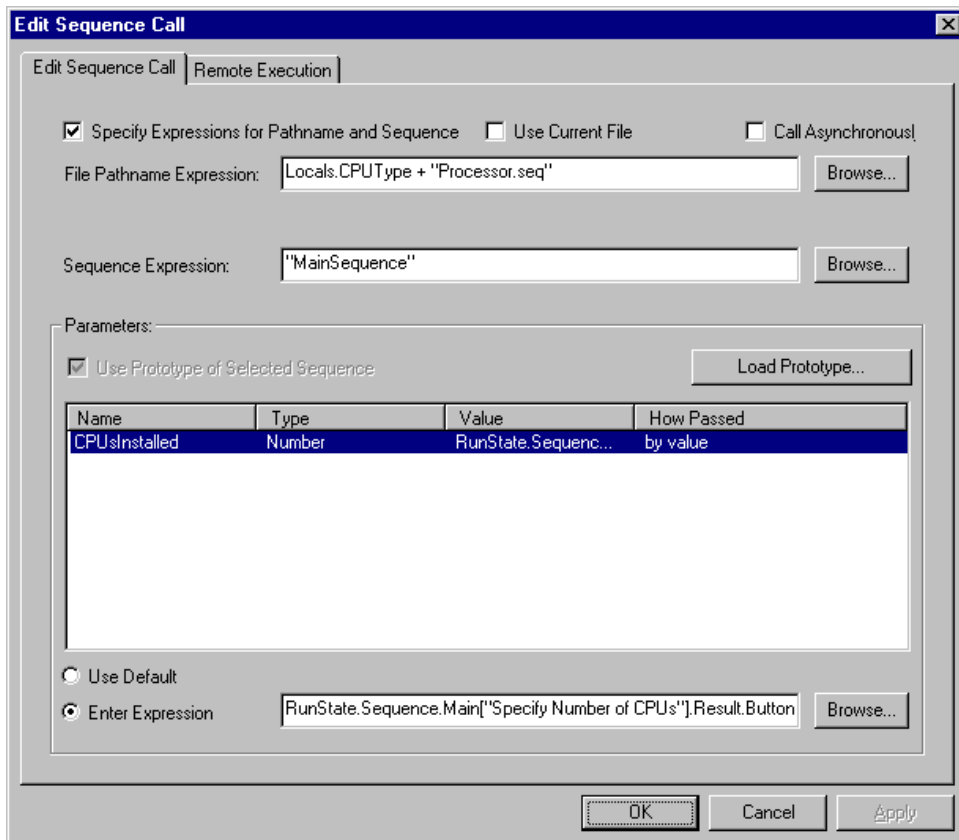


Figure 11-4. Dynamically Calling with an Expression

32. Click on **OK** to close the dialog box.
33. Select **File»Save As** and save the sequence in the `TestStand\Tutorial` directory as `Sample10.seq`.

Figure 11-5 shows the resulting sequence.

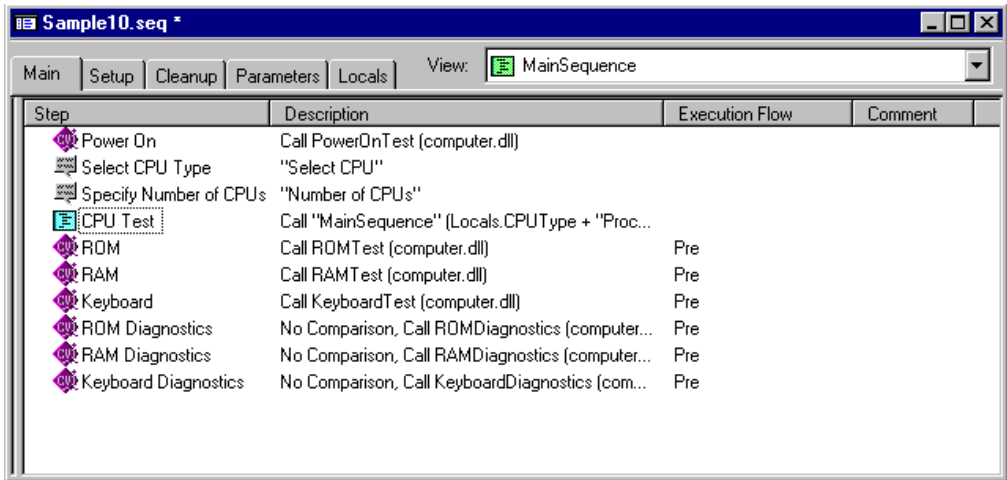


Figure 11-5. Dynamically Calling a Sequence

Running a Sequence

Complete the following steps to run a sequence dynamically:

1. Place a breakpoint on the CPU Test step by clicking to the left of the step icon or by right-clicking on the step and selecting **Toggle Breakpoint** from the context menu.
2. Select **Execute»Single Pass**.
3. Click on **Done** in the Test Simulator prompt.
4. Click on the **INTEL CPU** button in the Select CPU prompt.
5. Click on the **2** button in the Number of CPUs prompt.

- After the execution pauses at the breakpoint on the CPU Test step, single-step into the subsequence by selecting **Debug»Step Into**. Notice that the Call Stack pane lists `INTELProcessor.seq` at the bottom of the sequence call stack, as shown in Figure 11-6.

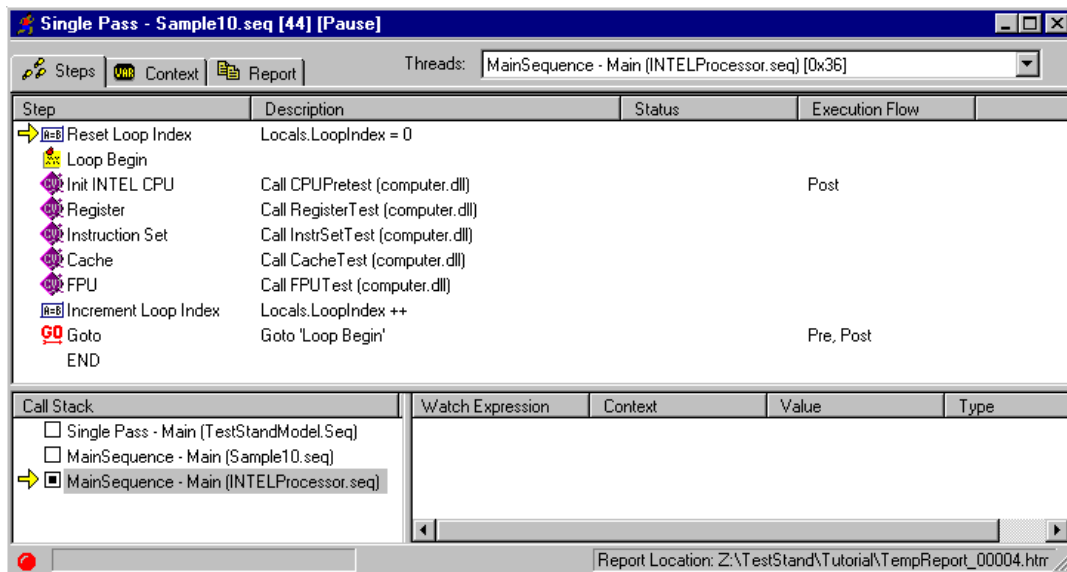


Figure 11-6. INTELProcessor.seq in the Call Stack Pane

- Click on the Context tab and notice the values of the two parameters for the sequence, as shown in Figure 11-7.

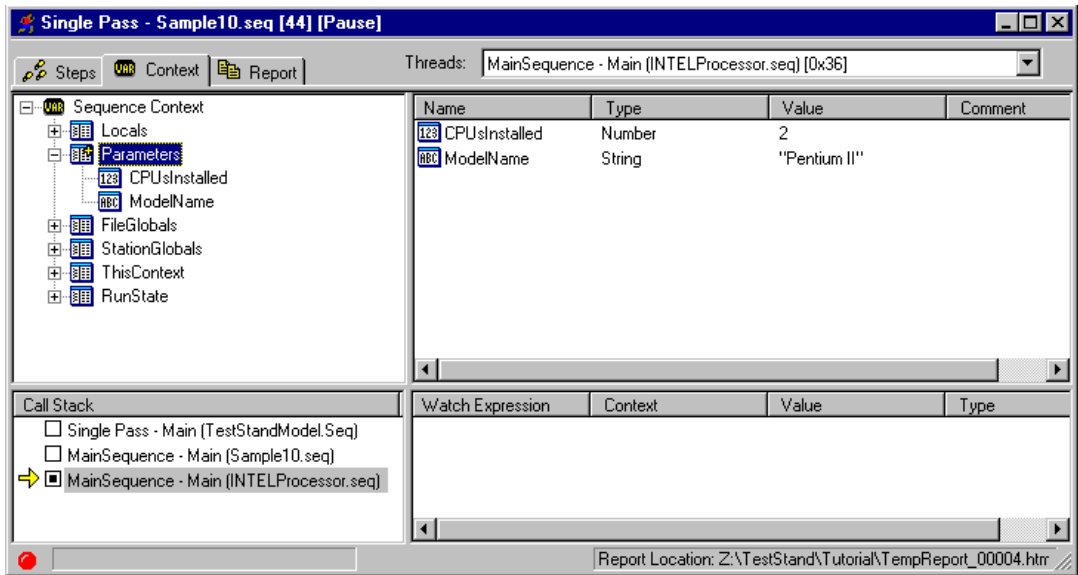


Figure 11-7. Sequence Parameters in the Context Tab

The value of the `CPUstnInstalled` parameter is equal to the value on the button you clicked on the Specify Number of CPUs prompt. Notice that `MainSequence` in the `INTELProcessor.seq` sequence file also requires a `ModelName` parameter. The sequence call step you created did not specify this parameter, so the engine initializes the parameter value to its default value.

- Complete the execution by selecting **Debug»Resume**.
- When the report file completes, review the report, but do not close the Execution window.
- Restart the execution by selecting **Execute»Restart**.
- Click on **Done** in the Test Simulator prompt.
- Click on the **AMD CPU** button in the Select CPU prompt.
- Click on the **3** button in the Number of CPUs prompt.
- When the execution pauses at the breakpoint on the CPU Test step, step into the subsequence by selecting **Debug»Step Into**.

Notice that the Call Stack pane lists `AMDProcessor.seq` at the bottom of the call stack.

15. Complete the execution by selecting **Debug»Resume**, and review the report.
16. Close the Execution window and the Sequence File window.

This concludes this tutorial session. In the next session, you learn how to customize the reports that TestStand generates.

Customizing the Report

In this chapter, you learn how to customize report generation within TestStand. Through the callback structure examined in Chapter 8, *Using Callbacks*, you can create your own Test Report Callback routine to develop reports in any format. Because changing report generation is so common, TestStand provides several options to configure the format of the test report without creating your own callback. You will examine each of these options in this chapter.

Setting Up the Example

Close all windows in the sequence editor so you can complete this tutorial session.

Configuring Test Report Options

1. Open `Sample1.seq` from the `TestStand\Tutorial` directory.

2. Select **Configure»Report Options**. The Report Settings dialog box appears, as shown in Figure 12-1.

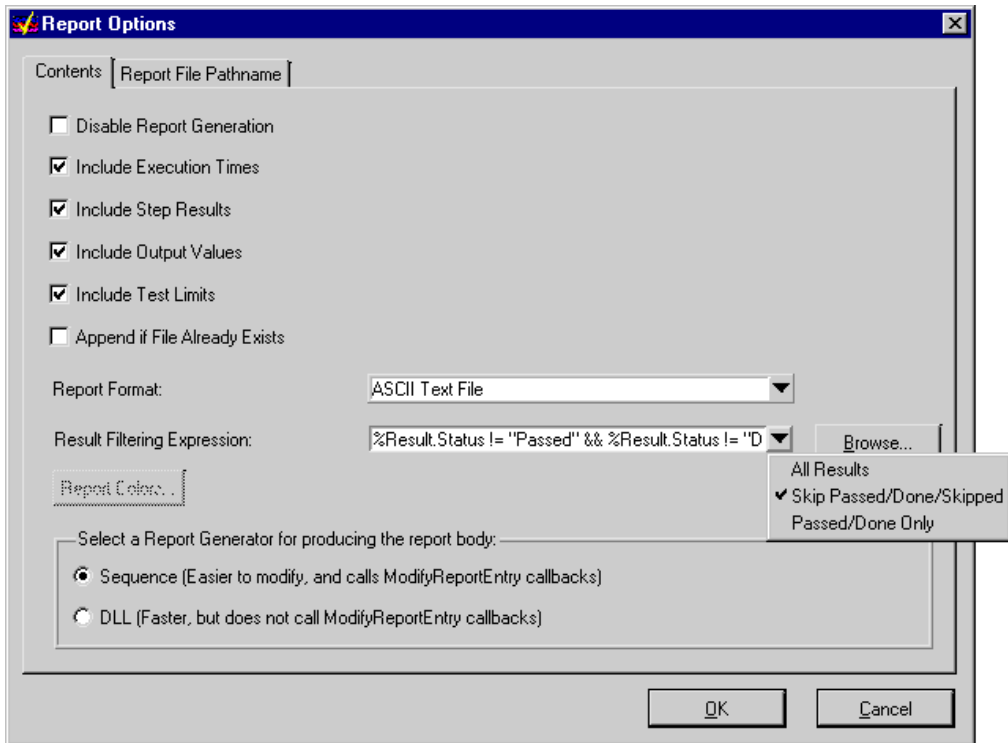


Figure 12-1. UUT Report Setting

3. Configure the test report options as shown in Figure 12-1.
 - a. Enable the following options:
 - Include Execution Times
 - Include Step Results
 - Include Result Values
 - Include Test Limits options
 - b. Set the Report Format Tag to ASCII Text File. This setting creates the test report in a standard ASCII format.
 - c. Set the Result Filtering Expression by clicking on the arrow to the right of the control and selecting Skip Passed/Done/Skipped. The Result Filtering Expression is used to determine the conditions that must be met for the results to be logged in the test report.

In this case, you are configuring it to record only the results of steps that do not pass or complete without any status. TestStand evaluates the expression at run-time when generating the report, and the results of each step is logged to the test report only if the expression is true.

4. Click on the Report File Pathname tab.

This tab allows you to configure the name and path for the test report file. You can, for example, create a new file for each UUT, or include the time and date in the name of the report file.

5. Leave the Report File Pathname tab options as they are and click on **OK** to close the dialog box.
6. Execute the sequence by selecting **Execute»Test UUTs**.
7. Run through several iterations of the sequence and select different components, other than the video, to fail.
8. Click on **Stop** in the UUT Information dialog box to stop the sequence execution.
9. Examine the test report and notice that the only step results that appear in the report are for steps that failed. The report format should appear similar to Figure 12-2.

```

*****
UUT Report
Serial Number:      12345
Date:              Monday, September 28, 1998
Time:              7:27:40 PM
Operator:          administrator
Execution Time:    3.854055 seconds
Number of Results: 8
UUT Result:       Failed
*****

Begin Sequence: MainSequence
(Z:\TestStand\Tutorial\Sample1.seq)

ROM:               Failed
  Module Time:     0.001427
End Sequence: MainSequence

End UUT Report

*****
UUT Report
Serial Number:      23456
Date:              Monday, September 28, 1998
  
```

Figure 12-2. Test Report in Text Format

10. Close the Execution window.
11. Select **Configure»Report Options**.
12. Change the Report Format Tag to Web Page.
13. Click on **OK** to close the dialog box.
14. Execute the sequence by selecting **Execute»Test UUTs**.
15. Run through several iterations of the sequence.
16. Click on **Stop** in the **UUT Information** dialog box to stop the sequence.
17. Examine the test report. The report format should appear as shown in Figure 12-3.

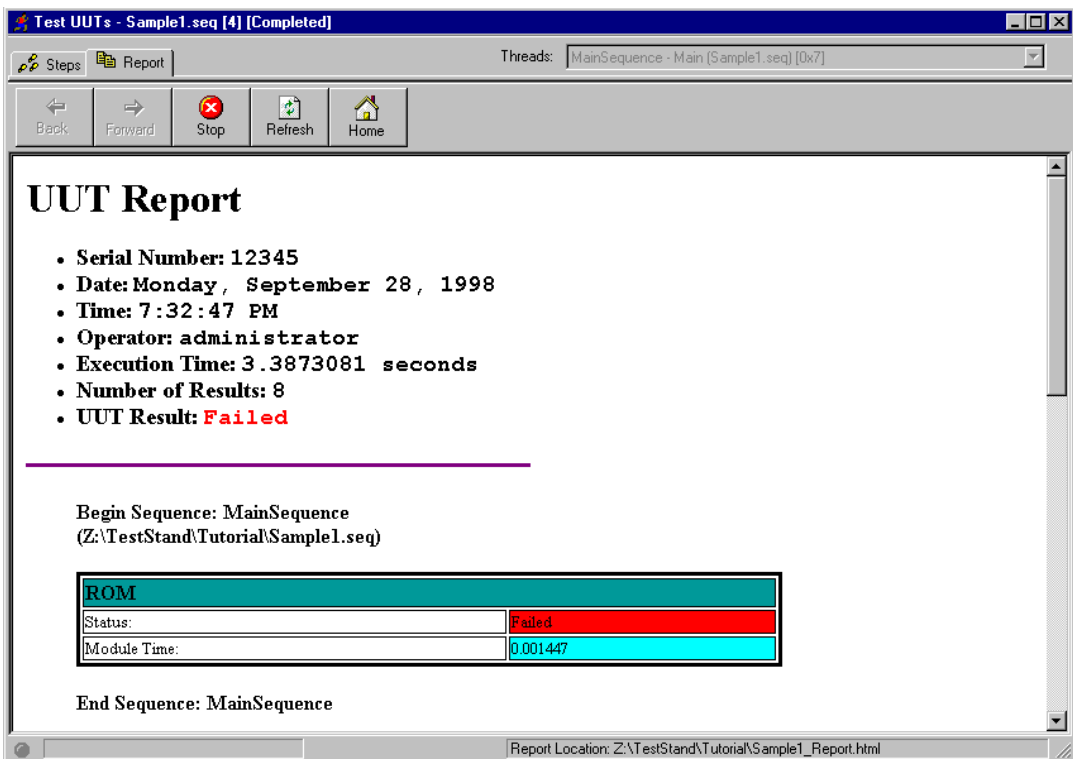


Figure 12-3. Test Report in HTML Format

18. Close the Execution window.
19. Select **Configure»Report Options**.

20. Set the Report Format Tag control back to ASCII Text File and set the Result Filtering Expression back to `True` by selecting the All Results option in the arrow pulldown list.
21. Click on **OK** to close the Report Options dialog box.

Using External Report Viewers

You might want to view the test report in external applications more suited for displaying and editing text, such as Microsoft Word or Microsoft Excel. TestStand refers to these external applications as external report viewers.

By default the Windows 95/NT/98 operating systems can associate an application with a file extension. For example, by default, Microsoft associates the `.doc` file extension with the Microsoft WordPad application. If you install Microsoft Word on your system, the Word installer replaces the WordPad file type association with itself for the `.doc` file type.

1. The TestStand process model automatically launches an external viewer for file types other than `.txt` and `.html`. To see this feature, complete the following steps:
 - a. Select the **Configure»Report Options**.
 - b. Select the Report File Pathname tab.
 - c. Disable the Use Standard Extension for Report Format option.
 - d. Enter `doc` in the Extension string control. TestStand will now create test reports with a `.doc` file extension.
 - e. Click on **OK** to close the Report Options dialog box.
2. Execute the sequence by selecting **Execute»Test UUTs**.
3. Run through several iterations of the sequence.
4. Select **Stop** in the **UUT Information** dialog box to stop the execution. TestStand now generates the text report and launches WordPad or Microsoft Word to display the test report.
5. Examine the test report and close the external report viewing application.
6. Change the report settings back as follows:
 - a. Select **Configure»Report Options**.
 - b. Change the Report Format Tag to Web Page on the Contents tab.
 - c. Enable the Use Standard Extension for Report Format option on the Report File Pathname tab.
 - d. Click on **OK** to close the dialog box.

If you want to define a file association independent of the operating system, you can configure this using **Configure»External Viewers**.

Adding to a Report

The default header for an HTML report appears as shown in Figure 12-3. You will now add a logo to the header of the HTML report using a report callback in the process model.

1. With the Sequence File window as the active window, select **Edit»Sequence File Callbacks** to open the `Sample1.seq` Callbacks dialog box.
2. Select the `ModifyReportHeader` callback.
3. Click on the **Add** button to add a callback to the sequence file.

Figure 12-4 shows the resulting dialog box.

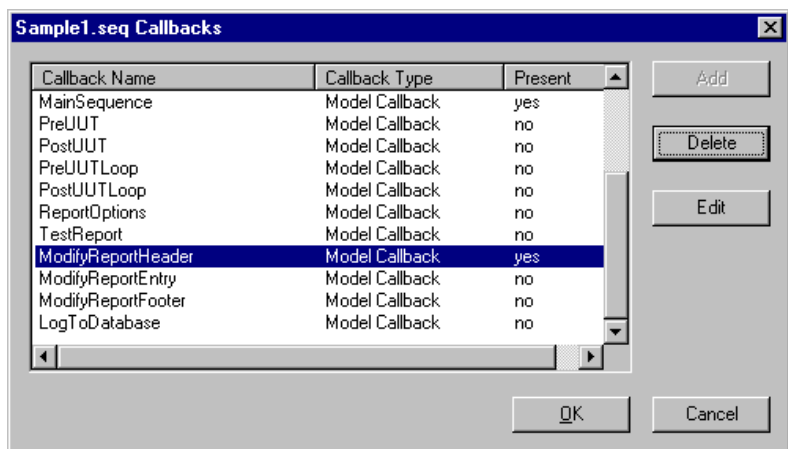


Figure 12-4. Callbacks Dialog Box

4. Click on the **Edit** button to close the dialog box and edit the new `ModifyReportHeader` callback sequence. Notice that the View pull-down ring lists the `ModifyReportHeader` as the active sequence, as shown in Figure 12-5.

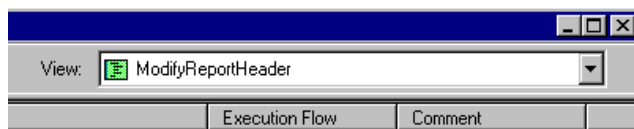


Figure 12-5. ModifyReportHeader in Sequence View

- Click on the Locals tab and right-click in the right pane to insert a local string variable, as shown in Figure 12-6.

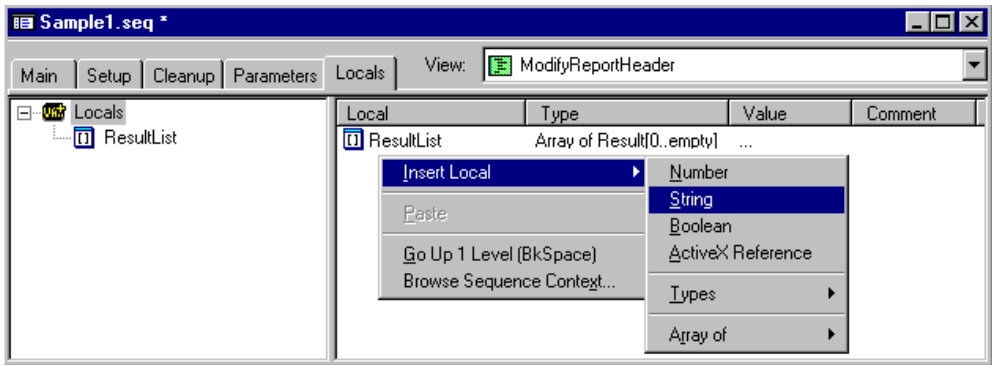


Figure 12-6. Insert String Local in ModifyReportHeader

- Rename the variable AddToHeader.
- Double-click on AddToHeader.
- Enter the following value in the Value field of the String Properties dialog box:

```
<IMG ALT='Logo Goes Here' SRC='Logo.jpg'><br><br>
<A HREF='http://www.natinst.com'>Visit Our Web
  Site</A><br>
```

- Click on **OK** to close the dialog box.
- Click on the Main tab to display the Main step group, which is empty.
- Right-click in the steps list and insert a Statement step by selecting **Insert Step»Statement** in the context menu.
- Rename this step Add Custom Logo.
- Right-click on the Add Custom Logo step and select **Edit Expression** from the context menu.
- Enter the following expression:

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

15. Click on **OK** to close the Edit Statement Step dialog box.
16. Select **File>Save As** and save the sequence in the TestStand\Tutorial directory as Sample11.seq. Figure 12-7 shows the resulting sequence file.

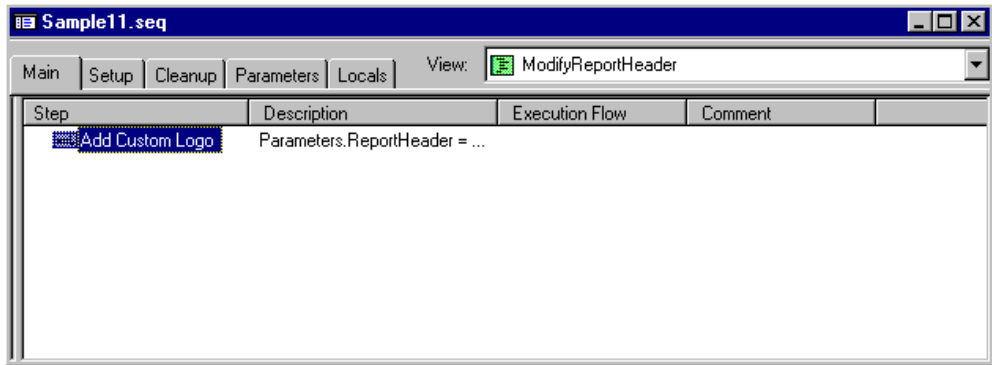


Figure 12-7. Completed ModifyReportHeader Sequence

17. Select **Execute>Single Pass**.
18. Click on **Done** in the Test Simulator prompt.

19. After the execution completes, view the report and notice the new logo image at the top of the UUT Report, as shown in Figure 12-8.

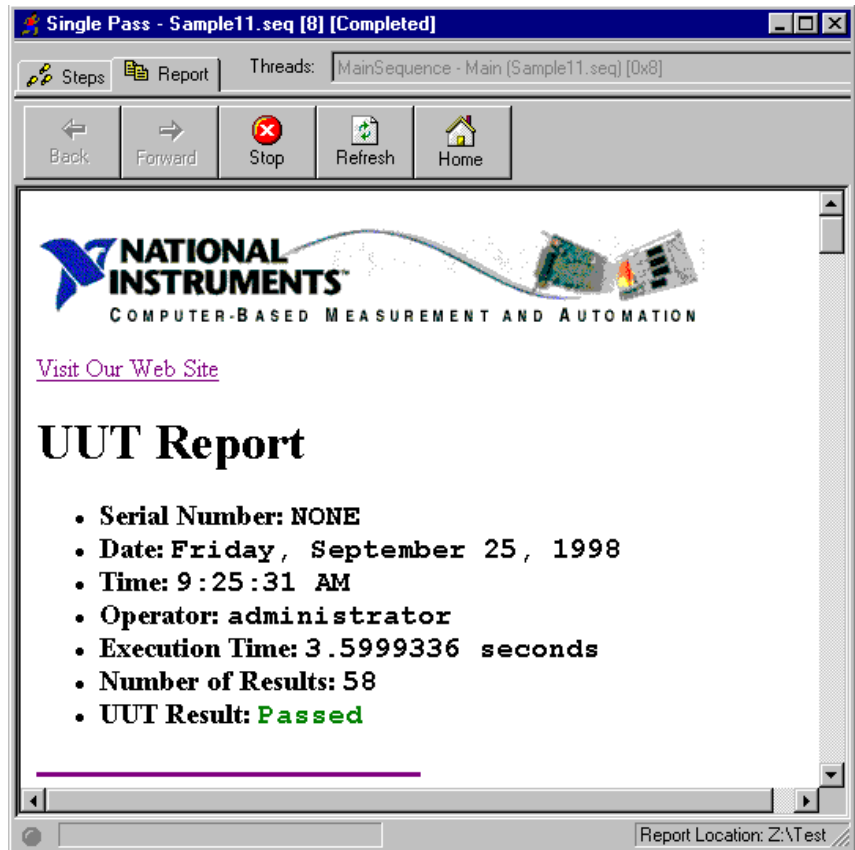


Figure 12-8. New HTML Header

20. Close the Execution window and the Sequence File window.

This concludes this final session in this tutorial. For more details on customizing reports, refer to Chapter 14, *Managing Reports*, in the *TestStand User Manual*.

Converting LabVIEW and LabWindows/CVI Test Executive Sequences

This chapter explains how to convert existing test sequences from Test Executive to TestStand. TestStand provides a conversion utility for converting LabVIEW Test Executive and LabWindows/CVI Test Executive sequence files to TestStand sequence files.

Converting LabVIEW Test Executive Sequences

To convert a LabVIEW Test Executive 5.1 sequence file to a TestStand sequence file, complete the following steps:

1. Select **Tools»Sequence File Converters»Convert LabVIEW Test Executive Sequence** in either the TestStand Sequence Editor or any of the execution interfaces.
2. In the file dialog box that appears, select a LabVIEW Test Executive sequence file to convert.
3. Enter a new file name for the converted test sequence.

TestStand displays a message indicating whether the conversion was successful.

Refer to the online help document in the `TestStand\Doc` directory, *Converting from the LabVIEW Test Executive to TestStand* (`LVTECompatibility.hlp`), for more information on converting sequences.

Note *If you are using version 4.0 or 5.0 of the LabVIEW Test Executive, you must first convert your sequence files to version 5.1 before switching to TestStand.*

Converting LabWindows/CVI Test Executive Sequences

To convert a LabWindows/CVI Test Executive 2.0.1 sequence file to a TestStand sequence file, complete the following steps:

1. Select **Tools»Sequence File Converters»Convert CVI Test Executive Sequence** in either the TestStand Sequence Editor or any of the run-time operator interfaces.
2. In the file dialog box that appears, select a LabWindows/CVI Test Executive sequence file to convert.
3. Enter a new file name for the converted test sequence.
TestStand displays a message indicating whether the conversion was successful.

Refer to the online help document in the `TestStand\Doc` directory, *Converting from the LabWindows/CVI Test Executive to TestStand* (`CVITECompatibility.hlp`), for more information on converting sequences.

Note *If the version of your LabWindows/CVI Test Executive is earlier than 2.0.1, you must first load and re-save the sequence file in the sequence editor for LabWindows/CVI Test Executive 2.0.1.*



Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422

Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as anonymous and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

support@natinst.com

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Québec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

Fax (____) _____ Phone (____) _____

Computer brand _____ Model _____ Processor _____

Operating system (include version number) _____

Clock speed _____ MHz RAM _____ MB Display adapter _____

Mouse ____ yes ____ no Other adapters installed _____

Hard disk capacity _____ MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

List any error messages: _____

The following steps reproduce the problem: _____

TestStand Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

Hardware revision _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

National Instruments software _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *Getting Started with TestStand*

Edition Date: December 1998

Part Number: 322017A-01

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

E-Mail Address _____

Phone (____) _____ Fax (____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meaning	Value
p-	pico	10^{-12}
n-	nano-	10^{-9}
μ -	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^3
M-	mega-	10^6
G-	giga-	10^9
t-	tera-	10^{12}

A

abort	To stop an execution without running any of the Cleanup step groups in the sequences on the call stack run. When you abort an execution, no report generation occurs.
active window	The window that user input affects at a given moment. The title of an active window is highlighted.
ActiveX reference property	A container of information that maintains a reference to an ActiveX object. TestStand maintains the value of the property as an <code>IDispatch</code> or <code>IUnknown</code> pointer.
ActiveX server	Any executable code that makes itself available to other applications according to the ActiveX standard. ActiveX implies a client/server relationship in which the client requests objects from the server and asks the server to perform actions on the objects.
Adapter	A service of the TestStand engine that steps use to invoke code in another sequence or in a code module. The adapter knows the type of the code module, how to call it, and how to pass parameters to it.
administrator	A user profile that usually contains all privileges for a test station.

Application Development Environment (ADE)	A programming environment such as LabVIEW, LabWindows/CVI, or Microsoft Visual C, in which you can create test modules and run-time execution operator interfaces.
Application Programming Interface (API)	A set of classes, methods, and properties that you use to control a specific service, such as the TestStand engine.
array property	A property that contains an array of single-valued properties of the same type.
ASCII	American Standard Code for Information Interchange.

B

block diagram	Pictorial description or representation of a program or algorithm. In LabVIEW, the block diagram which consists of executable icons called nodes and wires that carry data between the nodes, is the source code for the VI. The block diagram resides in the Diagram window of the VI.
breakpoint	An interruption in the execution of a program.
built-in property	A property that all steps or sequences contain. An example is the step run mode property. TestStand normally hides these properties in the sequence editor, although it lets you modify some of them through dialog boxes.
built-in step type property	A property that is common to all steps of the same type. A built-in step type property is either a class step type property or an instance step type property.
button	A dialog box item that, when selected, executes a command associated with the dialog box.

C

call stack	The chain of active sequences that are waiting for nested subsequences to complete.
callback	A sequence that is used to handle common tasks, such as serial number inquiry or report logging.
checkbox	A dialog box input that allows you to toggle between two possible options.

class	Defines a list of methods and properties that you can use with respect to the objects that you create as instances of that class. A class is like a data type definition except that it applies to objects rather than variables.
class step type property	A built-in step property that exists only in the step type itself. TestStand uses these properties to define how the step type works for all step instances. Step instances do not contain their own copies of class properties.
client sequence file	A sequence file that contains the main sequence a process model invokes to test a UUT. Each client sequence file contains a sequence called <code>MainSequence</code> . The process model defines what is constant about your testing process, whereas the client sequence file defines the steps that are unique to the different types of tests you run.
clipboard	A temporary storage area the operating system uses to hold text that is cut, copied, or deleted from a work area.
cluster	A set of ordered, unindexed data elements in LabVIEW of any data type including numeric, Boolean, string, array, or cluster. The elements must be all controls or all indicators.
code module	A program module, such as a Windows Dynamic Link Library (.dll) or LabVIEW VI (.vi), that contains one or more functions that perform a specific test or other action.
code template	A source file that contains skeleton code. The skeleton code serves as a starting point for the development of code modules for steps that use the step type.
configuration entry point	A sequence in the process model file that configures a feature of the process model. Configuration entry points usually save configuration information in a .ini file in the <code>TestStand\cfg</code> directory. By default, configuration entry points appear in the Configure menu. For example, the default process model contains the configuration entry point: <code>Config Report Options</code> . The <code>Config Report Options</code> entry point appears as Report Options in the Configure menu.
connector	Part of a LabVIEW VI or function node that contains its input and output terminals, through which data passes to and from the node.
container property	A property that contains no values, and typically contain multiple subproperties. Container properties are analogous to structures in C/C++ and to clusters in LabVIEW.

context menu	Menus accessed by clicking on an object. Menu options pertain to that object specifically.
control	An input and output device for entering data that appears on a panel or window.
control flow	The sequential order of instructions that determines execution order.
CPU	central processing unit
custom named data type	A data type that you define and name. For example, you might create a <code>Transmitter</code> data type that contains subproperties such as <code>NumChannels</code> and <code>PowerLevel</code> .
custom property	A property that you define in a step type. Each step you create with the step type has its own copy of the custom property. TestStand uses the value that you specify for the custom property in the step type as the initial value of the property in each new step you create. Normally, after you create a step, you can change the value of the property in the step.

D

developer	A user profile that usually contains all privileges associated with operating, debugging, and developing sequences and sequence files, but cannot configure user privileges, report options, or database options.
dialog box	A prompt mechanism in which you specify additional information needed to complete a command.
DLL	dynamic link library

E

Edit substep	A substep that the engine calls when editing the step. You invoke the substep with the menu item that appears in the context menu above Specify Module . The Edit substep displays a dialog box in which the sequence developer edits the values of custom step properties. For example, the Edit Limits item appears in the context menu for Numeric Limit test steps, and the Edit Pass/Fail Source item appears in the context menu for Pass/Fail test steps.
--------------	---

engine	<i>See</i> Test Executive Engine
engine callback	A sequence that TestStand invokes at specific points during execution. You use engine callbacks to tell TestStand to call certain sequences before and after the execution of individual steps, before and after interactive executions, after loading a sequence file, and before unloading a sequence file.
entry points	A sequence in the process model file that TestStand displays as a menu item, such as <code>Test UUTs</code> , <code>Single Pass</code> , and <code>Report Options</code> .
error occurred flag	A Boolean flag, <code>Step.Result.Error.Occurred</code> , that indicates whether a run-time error occurred in the step.
execution	An object that contains all the information TestStand needs to run a sequence, its steps, and any subsequences it calls. Typically, the TestStand sequence editor creates a new window for each execution.
execution entry point	A sequence in a process model that runs tests against a UUT. Execution entry points call the <code>MainSequence</code> callback in the client sequence file. The default process model contains two execution entry points: <code>Test UUTs</code> and <code>Single Pass</code> . By default, execution entry points appear in the <code>Execute</code> menu. Execution entry points appear in the menu only when the active window contains a sequence file that has a <code>MainSequence</code> callback.
execution pointer	A yellow pointer icon that shows the progress of execution by pointing to the currently executing step in the <code>Steps</code> tab.
Execution window	A window in the sequence editor that displays the steps an execution runs. When execution is suspended, the execution window displays the next step to execute and provides single-stepping options. You also can view variables and properties in any active sequence context in the call stack.
expression	A formula that calculates a new value from the values of multiple variable or properties. In expressions, you can access all variables and properties in the sequence context that is active when TestStand evaluates the expression. The following is an example of an expression: <code>Locals.MidBandFrequency = (Step.HighFrequency + Step.LowFrequency) / 2</code>

F

- front panel The interactive user interface of a LabVIEW VI. Modeled from the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, and other controls and indicators.
- front-end callback A common sequence that the sequence editor and run-time operator interfaces call. Front-end callbacks allow multiple applications to share the same implementation for a specific operation. TestStand installs the sequence file `FrontEndCallback.seq`, which contains the front-end callback sequence, `LoginLogout`.
- front-end callback sequence file A sequence file that contains front-end callbacks. TestStand installs the sequence file `FrontEndCallback.seq`, which contains the front-end callback sequence, `LoginLogout`.

G

- G The graphical programming language used to develop LabVIEW applications
- global variable TestStand defines two types of globals: sequence file globals and station globals. Sequence file globals are accessible by any sequence or step in the sequence file. Station globals are accessible by any sequence file loaded on the station. The values of station global variables are persistent across different executions and even across different invocations of TestStand.
- GUI *See* Run-Time Operator Interface.

H

- hex hexadecimal
- highlight The way in which input focus is displayed on a TestStand screen; to move the input focus onto an item.

I

in-process	When executable code runs in the same process space as the client, that is, an ActiveX server in a dynamic-link library(DLL).
instance step type property	A built-in step property that exists in each step instance. Each step that you create with the step type has its own copy of the property. TestStand uses the value you specify for an instance property in the step type as the initial value of the property in each new step that you create. Normally, after you create a step, you can change the values of its instance properties.
interactive mode	When you run steps by selecting one or more steps in a sequence and choosing the Run Selected Steps or Loop Selected Steps items in the context menu or menu bar. The selected steps in the sequence execute, regardless of any branching logic that the sequence contains. The selected steps run in the order in which they appear in the sequence.

K

kill	To stop a running, terminating, or aborting execution by terminating the thread of the execution without any cleanup of memory. This can leave TestStand in an unreliable state.
------	--

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench. A program development application based on the programming language G and used commonly for test and measurement purposes
list box	A dialog box item that displays a list of possible choices.
local variable	A property of a sequence that holds a value or additional subproperties. Only a step within the sequence can directly access the property value.

M

main sequence	The sequence that initiates the tests on a UUT. The process model invokes the main sequence as part of the overall testing process. The process model defines what is constant about your testing process, whereas main sequences define the steps that are unique to the different types of tests you run.
---------------	---

MB	megabytes of memory
menu bar	Horizontal bar that contains names of main menus.
method	Performs an operation or function on an object.
MFC	Microsoft Foundation Class Library
MHz	megahertz
model callback	A mechanism which allows a sequence file to customize the default behavior of a sequence in the process model.
model sequence file	A special type of sequence file that contains process model sequences. The sequences within the sequence file direct the high-level sequence flow of an execution when testing a UUT.
module adapter	A component that the TestStand engine uses to invoke code in another sequence or in a code module, such as LabVIEW. When invoking code in a code module, the adapter knows how to call it, and how to pass parameters to it.

N

named data type	A type of variable or property that you give a unique name. The data type usually contains multiple subproperties thus creating an arbitrarily complex data structure. All variables or properties that use the data type have the same data structure, but the values they contain can differ.
nested	Called by another step or sequence. If a sequence calls a subsequence, the subsequence is nested in the invocation of the calling sequence.
nested interactive execution	When you run steps interactively from an execution window for a normal execution that is suspended at a breakpoint. You can run steps only in the sequence and step group in which execution is suspended. The selected steps run within the context of the normal execution.
normal execution	When you start an execution in the sequence editor by selecting the Run Sequence Name item or one of the process model entry points from the Execute menu.
normal sequence file	Any sequence file containing sequences that test UUTs.
numeric property	A 64-bit floating-point value in the IEEE 754 format.

O

object	A service that an ActiveX server makes available to clients.
operator	A user profile that usually contains all privileges associated with operating a test station, but cannot debug sequence executions, edit sequence files, or configure user privileges, station options, report options, and database options.
out-of-process	When executable code does not run in the same process space as the client, such as an ActiveX server in an executable.

P

pop-up menus	<i>See</i> context menu
post actions	Actions that TestStand takes depending on the pass/fail status of the step or a custom condition the engine evaluates after executing a step. Post actions allow you to execute callbacks or jump to other steps after executing the step.
Post Step substep	A substep that the engine invokes after calling a step module. A Post Step substep might call a code module that compares the values the step module stored in step properties against limit values that the Edit substep stored in other step properties.
Pre Step substep	A substep that the engine invokes before calling the step module. For example, a Pre Step substep might call a code module that retrieves measurement configuration parameters and stores them into step properties for use by the step module.
preconditions	A set of conditions for a step that must be true for TestStand to execute the step during the normal flow of execution in a sequence.
process model	A series of operations before and after a test executive executes the sequence that performs the tests. Common operations include identifying the UUT, notifying the operator of pass/fail status, generating a test report, and logging results.
property	A container of information, which stores and maintains a setting or attribute of an object. A property can contain a single value, an array of values of the same type, or no value at all. A property can also contain any number of subproperties. Each property has a name.

property-array property A property containing a value that is an array of subproperties of a single type. In addition to the array of subproperties, property-array properties can contain any number of subproperties of other types.

R

RAM random-access memory

reference count Each ActiveX object keeps track of the number of things that reference it. This allows the object to decide when to free the resources it uses.

reference property *See* ActiveX reference property.

resource string Text strings stored in an external file so you can alter the strings without directly altering the application.

ROM read-only memory

root interactive execution When you run selected steps from a Sequence File window in an independent execution. Root interactive executions do not invoke process models.

run mode The mode in which you execute a step, such as normal, skip, force pass, force fail.

run-time error An error condition that forces an execution to terminate. When the error occurs while running a sequence, TestStand jumps to the Cleanup step group, and the error propagates to any calling sequence up through to the top-level sequence.

run-time operator interface A program that provides a graphical user interface for executing sequences at a production station. Sometimes the sequence editor and run-time operator interfaces are different aspects of the same program.

RTF rich text format

S

s seconds

sequence A series of steps that you specify for execution in a particular order. Whether and when a step is executed can depend on the results of previous steps.

sequence context	A TestStand object that contains references to all global variables and all local variables and step properties in active sequences. The contents of the sequence context changes depending on the currently executing sequence and step.
sequence editor	A program that provides a graphical user interface for creating, editing, and debugging sequences.
sequence file	A file that contains the definition of one or more sequences.
Sequence File window	A separate window within the sequence editor that a sequence file appears in.
sequence globals	Variables you can use to store data relevant to the entire sequence file. Each sequence and step in the sequence file can directly access these globals.
single-valued property	A property that contains a single value. TestStand has four types of these properties: Number properties, String properties, Boolean properties, and ActiveX reference properties.
source code template	A set of source files that contain skeleton code, which serves as a starting point for the development of code modules for steps. TestStand uses the code template when the sequence developer clicks on the Create Code button on the Source Code tab in the Specify Module dialog box for a step.
standard named data type	A data type that TestStand defines and names. You can add subproperties to the standard data types, but you cannot delete any of their built-in subproperties. The standard named data types are <code>Path</code> , <code>Error</code> , and <code>CommonResults</code> .
station callback sequence file	A sequence file that contains the station callback sequences. Station callbacks run before and after the engine executes each step in any normal or interactive execution.
station globals	Variables that are persistent across different executions and even across different invocations of the sequence editor or run-time operator interfaces. The TestStand engine maintains the value of station global variables in a file on the run-time computer.
station model	A process model that you select to use for all sequence files for a station. The TestStand installation program establishes <code>TestStandModel.seq</code> as the default station model file. You can use the Station Options dialog box to select a different station model.

step	Any action, such as calling a test module to perform a specific test, that you can include within a sequence of other actions.
step group	A set of steps in a sequence. A sequence contains the following groups of steps: Setup, Main, and Cleanup. When TestStand executes a sequence, the steps in the Setup group execute first, the steps in the Main group execute next, and the steps in the Cleanup group last.
step module	The code module that a step calls.
step property	A property of a step.
step result	A container property that contains a copy of the subproperties from the <code>Result</code> property of a step and additional execution information such as the name of the step and its position in the sequence. TestStand automatically creates a step result as each step executes and places the step result into a result list which TestStand uses to generate its reports.
step status	A string value that indicates the status of a step in an execution. Every step in TestStand has a <code>Result.Status</code> property. Although TestStand imposes no restrictions on the values to which the step or its code module can set the status property, TestStand and the built-in step types use and recognize a predefined set of values.
step type	A component that defines a set of custom step properties and standard behavior for each step of that type. All steps of the same type have the same properties, but the values of the properties can differ. Step types define their standard behaviors using substeps.
step-type-specific dialog box	A dialog box that step types display when their Edit substep is invoked. The dialog box lets you modify step properties that are specific to the step type. You invoke the dialog box with the menu item that appears in the context menu above Specify Module . For example, the Edit Limits item appears in the context menu for Numeric Limit test steps, and the Edit Pass/Fail Source item appears in the context menu for Pass/Fail test steps.
subsequence	A sequence that another sequence calls. You specify a subsequence call as a step in the calling sequence.
substep	Actions that a step type performs for a step besides calling the step module. You define a substep by selecting an adapter and specifying a module call. TestStand defines three different types of substeps: Edit substep, Pre Step substep, and Post Step substep.

substep module The code module that a Edit, Pre Step, or Post Step substep calls.

SVGA Super VGA

T

technician A user profile that usually contains all privileges associated with operating, and debugging sequences and sequences files, but cannot edit sequence files or configure user privileges, station options, report options, or database options.

template *See* code template.

terminal Object or region on a LabVIEW VI node through which data passes.

terminate To stop an execution by halting the normal execution flow, and running all the Cleanup step groups in the sequences on the call stack.

test executive engine A module or set of modules that provide an API for creating, editing, executing, and debugging sequences. A sequence editor or run-time execution operator interface uses the services of a test executive engine.

test module A code module that performs a test.

U

Unit Under Test (UUT) The device or component that you are testing.

user manager The component of the TestStand engine that maintains a list of users, their login names and passwords, and their privileges. You can access the user manager from the User Manager window in the sequence editor.

V

variables Properties that you can freely create in certain contexts. You can have variables that are global to a sequence file or local to a particular sequence. You can also have station global variables.

variables window A window that shows the values of all the currently active variables or properties.

VI	Virtual instrument.
VI library	Special file of type <code>.LLB</code> that contains a collection of related VIs for a specific use.

W

Watch window	A window that shows the values of user-selectable variables and expressions that are currently active.
window	A working area that supports specific tasks related to developing and executing programs.
wire	Tool used to define data paths between source and sink terminals.