

COMPREHENSIVE SERVICES

We offer competitive repair and calibration services, as well as easily accessible documentation and free downloadable resources.

SELL YOUR SURPLUS

We buy new, used, decommissioned, and surplus parts from every NI series. We work out the best solution to suit your individual needs.

 Sell For Cash  Get Credit  Receive a Trade-In Deal

OBSOLETE NI HARDWARE IN STOCK & READY TO SHIP

We stock **New**, **New Surplus**, **Refurbished**, and **Reconditioned** NI Hardware.



Bridging the gap between the manufacturer and your legacy test system.

 1-800-915-6216

 www.apexwaves.com

 sales@apexwaves.com

All trademarks, brands, and brand names are the property of their respective owners.

Request a Quote

 **CLICK HERE**

VXI-MXI

LabWindows™ /CVI™

Getting Started with LabWindows/CVI

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 450 510 3055,
Canada (Toronto) 905 785 0085, Canada (Vancouver) 604 685 7530, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11,
France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 51190000, Israel 972 0 3 6393737,
Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9131 0918,
Mexico 01 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60,
Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 65 6226 5886,
Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00,
Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227, Thailand 662 992 7519,
United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

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 document 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 THEREOF 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.

In regards to components used in USI (Xerxes C++, ICU, and HDF5), the following copyrights apply. For a listing of the conditions and disclaimers, refer to the [USICopyrights.chm](http://www.usi.com/copyrights.htm).

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Copyright © 1999 The Apache Software Foundation. All rights reserved.

Copyright © 1995–2003 International Business Machines Corporation and others. All rights reserved.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities

Copyright 1998, 1999, 2000, 2001, 2003 by the Board of Trustees of the University of Illinois. All rights reserved.

Trademarks

CVT™, DataSocket™, DIAdem™, HS488™, IVI™, MXI™, NAT4882™, National Instruments™, NI™, ni.com™, NI-488.2™, NI-DAQ™, NI-VISA™, NI-VXI™, TNT4882C™, and VXIpc™ are trademarks of National Instruments Corporation.

FireWire® is the registered trademark of Apple Computer, Inc. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or ni.com/patents.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN

COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS. THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions	ix
Related Documentation.....	x

Chapter 1

Introduction to LabWindows/CVI

Installing LabWindows/CVI.....	1-1
Learning About LabWindows/CVI	1-2
LabWindows/CVI System Overview	1-3
LabWindows/CVI Program Development Overview	1-3
Using C in LabWindows/CVI	1-4
LabWindows/CVI Program Structure	1-4

PART I

Getting Acquainted with the LabWindows/CVI Development Environment

Chapter 2

Loading, Running, and Editing Source Code

Setting Up	2-1
Loading a Workspace into LabWindows/CVI	2-1
Workspace Window	2-2
Standard Input/Output Window	2-4
Source Window	2-4
Editing Tools.....	2-5
Operating Projects with a User Interface	2-7

Chapter 3

Interactive Code Generation Tools

Setting Up	3-1
Library Tree.....	3-1
Accessing the User Interface Library	3-3
Function Panel Fundamentals.....	3-4
Function Panel Controls	3-4
Function Panel Help	3-4
Drawing a Graph	3-5

Inserting Code from a Function Panel	3-6
Analyzing Data	3-7
Output Values on a Function Panel.....	3-8
Recalling a Function Panel	3-9
Finishing the Program.....	3-9
Interactively Executing a Function Panel.....	3-10

Chapter 4

Executing and Debugging Tools

Setting Up.....	4-1
Step Mode Execution	4-1
Breakpoints.....	4-3
Predetermined Breakpoints	4-3
Instant Breakpoints	4-4
Displaying and Editing Data	4-5
Variables Window.....	4-5
Array Display	4-7
String Display	4-8
Watch Window	4-8
Tooltips	4-9
Graphical Array View	4-10

PART II

Building an Application in LabWindows/CVI

Chapter 5

Building a Graphical User Interface

User Interface Editor	5-1
Source Code Connection	5-1
CodeBuilder.....	5-2
Sample Project.....	5-2
Setting Up	5-2
Building a User Interface Resource (.uir) File.....	5-2

Chapter 6

Using Function Panels and Libraries

Setting Up.....	6-1
Analyzing the Source Code.....	6-1

Generating a Random Array of Data.....	6-3
Building the PlotY Function Call Syntax	6-3
Running the Completed Project.....	6-5

Chapter 7

Adding Analysis to Your Program

Setting Up	7-1
Modifying the User Interface	7-1
Writing the Callback Function	7-3
Running the Program.....	7-5

Chapter 8

Using an Instrument Driver

Setting Up	8-1
Loading the Instrument Driver	8-1
Using the Instrument Driver	8-2
Interactive Function Panel Execution.....	8-3
Initializing the Instrument	8-3
Configuring the Instrument	8-3
Reading Data with an Instrument Driver.....	8-4
Closing the Instrument	8-5
Running the Program	8-6
Adding the Instrument to Your Project	8-8

Chapter 9

Additional Exercises

Base Project	9-1
Exercise 1: Adding a Channel Control	9-2
Assignment.....	9-2
Exercise 2: Setting User Interface Attributes Programmatically.....	9-3
Assignment.....	9-4
Exercise 3: Storing the Waveform on Disk	9-4
Assignment.....	9-4
Exercise 4: Using Pop-up Panels	9-5
Assignment.....	9-6
Exercise 5: Adding User Interface Events	9-7
Assignment.....	9-8
Exercise 6: Timed Events	9-8
Assignment.....	9-8

PART III

Instrument Control and Data Acquisition

Chapter 10

Getting Started with GPIB and VXI Instrument Control

Getting Started with the GPIB Controller	10-1
Introduction to GPIB.....	10-1
Installing the GPIB Interface Board.....	10-2
Configuring the GPIB Driver Software	10-2
Configuring LabWindows/CVI for GPIB.....	10-2
Developing Applications.....	10-2
Getting Started with the VXI Controller	10-3
Introduction to VXI.....	10-3
VXI Development System	10-3
Installing and Configuring VXI Hardware	10-3
Configuring VXI Driver Software	10-4
Configuring LabWindows/CVI for VXI.....	10-4
Developing Applications.....	10-4
Using Instrument Drivers	10-4

Chapter 11

Getting Started with Data Acquisition

Introduction to Data Acquisition	11-1
Installing Software.....	11-1
Configuring Your Device for Data Acquisition	11-2
Testing the Operation of the Device and Configuration	11-2
Developing Applications	11-2
DAQ Assistant	11-2
Data Acquisition Library Sample Programs	11-3
Related Documentation	11-3

Appendix A

Technical Support and Professional Services

Glossary

Index

About This Manual

Getting Started with LabWindows/CVI is a hands-on introduction to the LabWindows™/CVI™ software package. This manual is intended for first-time LabWindows/CVI users. To use this manual effectively, you should be familiar with DOS, Microsoft Windows, and the C programming language.

Conventions

The following conventions appear in this manual:

» The » symbol leads you through nested menu items or dialog box options to a final action and also through the Table of Contents in the *LabWindows/CVI Help* to a help topic. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

This symbol also leads you through the LabWindows/CVI Library Tree to a function panel. For example, **User Interface Library»Pop-up Panels»InstallPopup** directs you to expand the User Interface Library in the Library Tree, expand Pop-up Panels, and select InstallPopup.



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



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

bold

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

italic

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

monospace

Text in this font denotes text or characters that you should 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.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- Harbison, Samuel P. and Guy L. Steele, Jr. *C: A Reference Manual*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1995.
- *LabWindows/CVI Help*
- *LabWindows/CVI Quick Reference*
- *LabWindows/CVI Instrument Driver Developers Guide*
- *LabWindows/CVI Bookshelf*
- *LabWindows/CVI Release Notes*
- *NI-DAQmx Help*
- *DAQ Quick Start Guide*
- *DAQ Assistant Help*
- *Traditional NI-DAQ Function Reference Help*
- *NI-VISA Help*
- *NI-488.2 Help*

Introduction to LabWindows/CVI

This chapter contains an overview of the LabWindows/CVI software development system, a program development overview, and an overview of the LabWindows/CVI documentation set.

Installing LabWindows/CVI

To install LabWindows/CVI, follow the installation instructions in the *LabWindows/CVI Release Notes* that come with your package. Table 1-1 lists the LabWindows/CVI components and their location.

Table 1-1. LabWindows/CVI Components

Location	Components
\bin	LabWindows/CVI library files and help files
\extlib	Files for using the LabWindows/CVI libraries with external compilers
\fonts	Font files required for graphics operations
\include	Include files associated with libraries
\instr	Sample instrument modules
\redist	Files required to redistribute your programs
\samples	Source code to example programs
\sdk	Software Development Kit (SDK) library files
\toolslib	Additional development tools and libraries
\tutorial	Programs used in the tutorial exercises throughout this manual
\vxd	VxD sample code templates
\wizard	Files used in LabWindows/CVI wizards

If you want to install LabWindows/CVI on a network, contact National Instruments for licensing information.

Learning About LabWindows/CVI

The following method is only one way to learn about LabWindows/CVI.

1. Read the remainder of this chapter to learn about the concepts and capabilities of LabWindows/CVI.
2. Complete the tutorial exercises in Chapters 2 through 9. Each section of the tutorial builds on previous sections.
3. Review the example programs and the tutorial programs included in the LabWindows/CVI installation. These programs illustrate many of the new features in LabWindows/CVI and the concepts discussed in this manual. Use the example programs as a starting point for your programs. You can search for example programs using NI Example Finder, accessible through **Help»Find Examples**.

As you work through the tutorial, refer to the LabWindows/CVI documentation set for more information about the concepts presented in this manual.

- *LabWindows/CVI Quick Reference*—Contains an overview of the steps involved in creating a LabWindows/CVI program and shows the LabWindows/CVI libraries and classes
- *LabWindows/CVI Bookshelf*—Provides a comprehensive list of documentation, including links to application notes and white papers
- *LabWindows/CVI Instrument Driver Developers Guide*—Describes developing and adding instrument drivers to the LabWindows/CVI Instrument Library
- *LabWindows/CVI Help*—Contains the following sections:
 - *Using LabWindows/CVI*—Information about windows, menus, commands, dialog boxes, and options for customizing configuration defaults
 - *Library Reference*—Reference information for all LabWindows/CVI library functions
 - *Programmer Reference*—Information about developing programs in LabWindows/CVI
 - *Tools Library*—Descriptions of the additional instrument drivers included with LabWindows/CVI

If you are new to LabWindows/CVI, complete the tutorial exercises in this manual first. Then you are ready to read *Using LabWindows/CVI* in the *LabWindows/CVI Help*. These two pieces of documentation contain the fundamental information that you need to get started. For a listing of documentation and the location of specific LabWindows/CVI information, refer to **Start»Programs»National Instruments»LabWindows CVI»LabWindows CVI Bookshelf**.

LabWindows/CVI System Overview

LabWindows/CVI is a software development environment for C programmers. You can use LabWindows/CVI for the following tasks:

- Interactively develop programs
- Access powerful function libraries for creating data acquisition and instrument control applications
- Take advantage of a comprehensive set of software tools for data acquisition, analysis, and presentation

You can edit, compile, link, and debug ANSI C programs in the LabWindows/CVI development environment. Use the functions in the LabWindows/CVI function libraries to write programs. In addition, each function has an interface called a *function panel* in which you can execute the function and generate code for calling the function. While you work in function panels, you can right-click the panel or a control to access help for the function, control, function class, and function library.

The power of LabWindows/CVI lies in its libraries. The libraries contain functions for developing all phases of your data acquisition and instrument control system.

- **Data Acquisition**—IVI Library, GPIB/GPIB 488.2 Library, NI-DAQmx Library, Traditional NI-DAQ Library, RS-232 Library, VISA Library, VXI Library, NI-CAN Library
- **Data Analysis**—Formatting and I/O Library, Analysis Library, optional Advanced Analysis Library
- **Data Presentation**—User Interface Library
- **Networking and Interprocess Communication Applications**—Dynamic Data Exchange (DDE) Library, Transmission Control Protocol (TCP) Support Library, ActiveX Library, Internet Library, DIAdem Connectivity Library, DataSocket Library

In addition, you can access a complete standard ANSI C Library within the LabWindows/CVI development environment.

LabWindows/CVI Program Development Overview

While you work with LabWindows/CVI, adhere to the same good programming practices common to all languages and development environments. For example, it is a good idea to create a functional design of your program before you begin writing code. Also, maintain good documentation and comments in your code to help you better manage program development.

User Interface

With the LabWindows/CVI User Interface Editor, you can build complex, interactive panels for programs with minimum effort. During the process of graphical user interface (GUI) design, developers often decide how they want their programs to acquire and display data and how they want menus, panels, controls, and dialog boxes to behave. Therefore, the user interface is a natural place to begin the design of a program.

The User Interface Library contains functions to control GUIs from application programs. LabWindows/CVI provides a User Interface Editor where you create GUIs. You can create GUI panels that contain graphs, strip charts, and other controls. You also can create pull-down menus, display graphic images, and prompt users for input with pop-up dialog boxes. You can use the User Interface Editor to create these items, or you can use the User Interface Library to create these items programmatically.

To learn more about the elements of the user interface and the functions that you can use to connect your interface to the rest of your program, refer to the *Using LabWindows/CVI» Developing a Graphical User Interface* and the *Library Reference»User Interface Library* sections of the *LabWindows/CVI Help*.

Program Shell Generation with CodeBuilder

After you design a GUI in the User Interface Editor, you can use CodeBuilder to automatically generate a program shell based on the components in the GUI. CodeBuilder writes code for all control callback functions and creates a program skeleton that loads and displays GUI windows at program startup. CodeBuilder saves development time by automating many of the common coding tasks required for writing a Windows program. Activities later in this tutorial introduce you to CodeBuilder.

Program Control

The program control portion of the program coordinates data acquisition, data analysis, and the user interface. Program control contains the control logic for managing the flow of program execution and user-defined support functions.

You must write most of the code that controls your LabWindows/CVI program. You can study the code in the LabWindows/CVI example programs to better understand how to write code for controlling your program. Pay attention to the use of callback functions in the example programs. Callback functions can help you control the flow of applications.

Data Acquisition

Normally programs must control the acquisition of data from an instrument or from a plug-in data acquisition (DAQ) device. The other portions of the program can analyze and display that data.

The various LabWindows/CVI libraries provide functions for creating the data acquisition program element. Use the library functions to control GPIB, RS-232, and VXI devices and National Instruments DAQ devices.

GPIB functions are introduced in *Library Reference»GPIB/GPIB 488.2 Library* of the *LabWindows/CVI Help*, with detailed function descriptions available in the *NI-488.2 Help* that comes with your GPIB interface. VXI Library functions are documented in the *NI-VXI API Reference Help* that comes with your VXI controller.

National Instruments provides two DAQ application programming interfaces (APIs). Traditional NI-DAQ is an upgrade of version 6.9.x of NI-DAQ. Traditional NI-DAQ has the same functions and API and works in the same way as NI-DAQ 6.9.x, but Traditional NI-DAQ is updated so you can use Traditional NI-DAQ and NI-DAQmx on the same computer. NI-DAQmx is the next-generation version of the NI-DAQ driver with new functions and development tools for controlling your measurement devices. To determine which version of NI-DAQ you should install, refer to the *DAQ Quick Start Guide*, located in **Start»Programs»National Instruments»NI-DAQ**.

If you have installed NI-DAQmx, you can refer to *Library Reference»NI-DAQmx Library* in the *LabWindows/CVI Help* for a description of NI-DAQmx Library functions. If you need documentation about which Traditional NI-DAQ functions apply to specific DAQ devices along with specific information about using those functions in LabWindows/CVI, refer to the *Traditional NI-DAQ Function Reference Help*, located in **Start»Programs»National Instruments»NI-DAQ**.

With LabWindows/CVI, you can choose from a variety of drivers for GPIB, RS-232, and VXI instruments. Refer to *Using LabWindows/CVI»Instrument Drivers* in the *LabWindows/CVI Help*. Refer to Chapter 10, *Getting Started with GPIB and VXI Instrument Control*, and Chapter 11, *Getting Started with Data Acquisition*, of this manual for more information about these topics.

Data Analysis

After you acquire data, you must analyze the data. For example, you might want to perform formatting, scaling, signal processing, statistical analysis, and curve fitting. The following libraries contain functions that perform these operations:

- Formatting and I/O Library
- Analysis Library (available in Base package)
- Advanced Analysis Library (available in Full Development System package)

You can access function descriptions, function trees, and general information about these libraries in the *LabWindows/CVI Help*.

Getting Acquainted with the LabWindows/CVI Development Environment

- Chapter 2, *Loading, Running, and Editing Source Code*, describes how to load and run projects in the LabWindows/CVI development environment. You will learn about some of the windows in LabWindows/CVI, the different types of files that you can use in a LabWindows/CVI project, some of the source code editing techniques available in LabWindows/CVI, and how to use projects in LabWindows/CVI.
- Chapter 3, *Interactive Code Generation Tools*, describes tools available for interactive code generation in LabWindows/CVI.
- Chapter 4, *Executing and Debugging Tools*, describes tools available for executing and debugging in the LabWindows/CVI interactive program. This chapter describes the step modes of execution, breakpoints, the Variables window, the Array Display, the String Display, and the Watch window.

Loading, Running, and Editing Source Code

In this chapter, you will load and run projects in the LabWindows/CVI development environment and learn about the following topics:

- Principal windows in the LabWindows/CVI development environment
- Types of files that you can include in a LabWindows/CVI project
- Useful source code editing techniques

Setting Up

Launch LabWindows/CVI by selecting **Start»Programs»National Instruments»LabWindows CVI x.x»NI LabWindows CVI**. When you open LabWindows/CVI, you see an empty Workspace window.



Note Instead of using LabWindows/CVI commands to manipulate windows—close, maximize, minimize, or position—you can use any of the windowing methods that are standard in the Windows operating system.

If more than one person will work on the Getting Started tutorial, be sure to save original, unchanged copies of the example files for each person.



Note The exercises in this tutorial refer to shortcut keys. You can customize shortcut keys by selecting **Options»Change Shortcut Keys**. This tutorial refers to the default shortcut keys.

Loading a Workspace into LabWindows/CVI

To view some of the editing and execution features of the LabWindows/CVI development environment, load a workspace into the LabWindows/CVI Workspace window.

1. Select **File»Open**. The **Open** submenu shows the different file types that you can create and edit in LabWindows/CVI.
2. Select **Workspace (*.cws)** and load `tutorial.cws` from the `\tutorial` subdirectory.

After you load `tutorial.cws`, the Workspace window appears as shown in Figure 2-1.

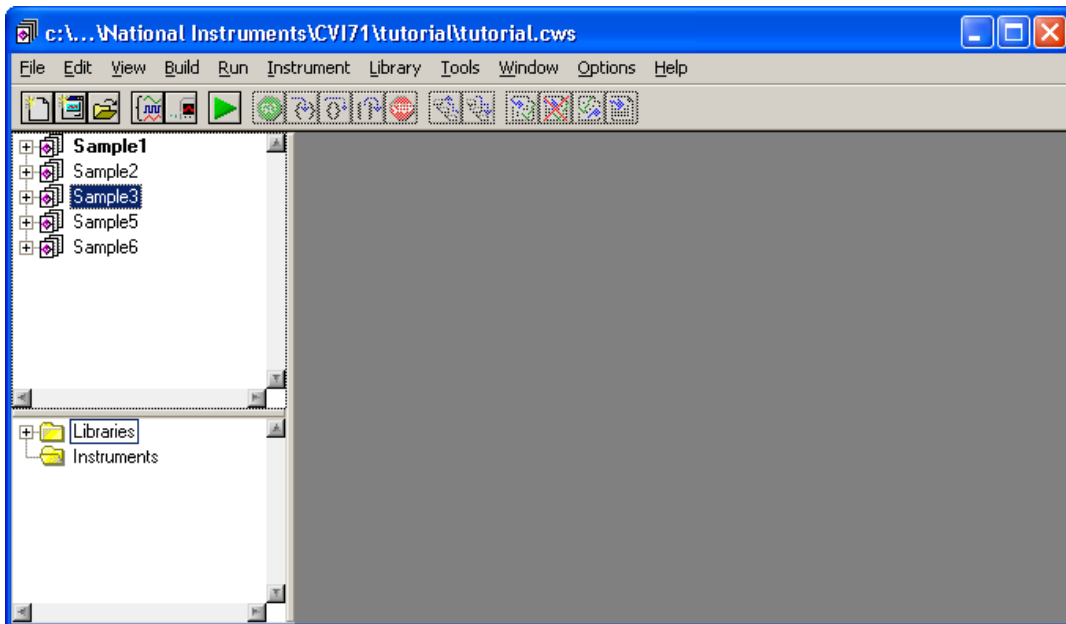


Figure 2-1. tutorial.cws in the Workspace Window



Note This manual introduces you to many LabWindows/CVI windows, utilities, and tools. For more information about each menu item, dialog box, and window, refer to *Using LabWindows/CVI* in the *LabWindows/CVI Help*.

The following sections introduce the Workspace window, the Standard Input/Output window, and the Source window.

Workspace Window

The Workspace window consists of five areas—the Project Tree, the Library Tree, the Window Confinement Region, the Debugging Region, and the Output Region.

- **Project Tree**—Contains the list of files in each project in the workspace. Right-click the different elements of the Project Tree to see the list of options available for files and folders.
- **Library Tree**—Contains a tree view of the functions in LabWindows/CVI libraries and instruments. You can arrange the library functions in alphabetical order, by function name or function panel title, or in a flat list instead of a hierarchical class structure.
- **Window Confinement Region**—Contains open Source windows, User Interface Editor, Function Tree Editor, and function panels. If you want to work with these windows outside of the Window Confinement Region, select **Window»Release Window**.

- **Debugging Region**—Contains the Variables, Watch, and Memory windows. Use these windows to view variable values and program memory during debugging.
- **Output Region**—Contains the Build Errors, Run-Time Errors, Source Code Control Errors, Debug Output, and Find Results windows. These windows contain lists of errors, output, and search matches.

The workspace you use throughout this tutorial contains five projects. To work on a project, right-click the project you want to work on and select **Set Active Project** from the context menu. The active project name is bold. You can view status information for the files in the projects. To view status information, select **View»Columns** and select the information you want to view. Figure 2-2 shows how this information appears in the Project Tree.

Project(s)	Status	Date/Time	Size
Sample1		5/19/2004, ...	3 KB
Source Files			
sample1.c	C	5/6/2004, 1...	1 KB
Sample2		5/19/2004, ...	4 KB
Source Files			
sample2.c		5/6/2004, 1...	6 KB
Include Files			
sample2.h		5/6/2004, 1...	2 KB
User Interface Files			
sample2.uir		5/6/2004, 1...	4 KB
Sample3		5/19/2004, ...	3 KB
Source Files			
sample3.c	C	5/6/2004, 1...	1 KB

Figure 2-2. Status Columns in the Project Tree

To add existing files to projects—C source files, header files, object modules, DLLs, C libraries, user interface files, instrument drivers—select **Edit»Add Files to Project**. You also can drag and drop the file onto the Project Tree.

The Workspace and Source windows, User Interface Editor, Function Tree Editor, and function panels have optional toolbars for quick access to many of the editing and debugging features in LabWindows/CVI. If you are unsure of what a particular toolbar icon represents, place your cursor over the icon to view the built-in tooltip help. To customize the icons in your toolbar, select **Options»Toolbar**.

Running the Project

To run the Sample 1 project, right-click **Sample1** and select **Set Active Project**. Select **Run»Debug sample1_dbg.exe**. LabWindows/CVI responds to the **Run** command by completing the following tasks:

- Compiling any source files in the project
- Linking the project with the libraries used

- Executing the compiled code
- Turning off the “C” indicator in the Status column after the source compiles
- Displaying the word *Running* in the upper left-hand corner of the Workspace window while the project runs

The Sample 1 project is a program that generates 100 random numbers and outputs them to the Standard Input/Output window in LabWindows/CVI.

Error Messages

If the compiler finds an error during the compiling or linking process, the Build Errors or Run-Time Errors window appears in the Output Region. The error windows list the number of errors LabWindows/CVI detects in each source file and a description of the errors. For example, if you get an illegal character error or a syntax error, LabWindow/CVI opens the Build Errors window. The line number of the error appears to the left of the error type. You can double-click the error to highlight the line in the file in which the error occurs. Correct the error and rerun your program.

To close the error window, click the **x** box located in the upper right-hand corner of the window. Select **Window»Build Errors** to open the window.

Standard Input/Output Window

The Standard Input/Output window is where simple, text-based information is displayed to or received from the user during program execution. When you use the ANSI C `stdio` library to develop your C programs in LabWindows/CVI, the results of the `printf` and `scanf` functions appear in the Standard Input/Output window.

Source Window

Use the Source window in LabWindows/CVI to develop C source files for projects. After running the Sample 1 project, close the Standard Input/Output window by pressing <Enter>. Double-click `sample1.c` in the Project Tree to open the source code in a Source window. As you can see, the source code for Sample 1 contains standard ANSI C-compatible code.

You can use the menu items in the Source window to edit files, debug code, compile files, make an installer application for programs, and so on. For more information about the Source window, refer to *Using LabWindows/CVI»Writing Source Code* in the *LabWindows/CVI Help*.

The Source window is compatible with the full ANSI C language specification. You can use any ANSI C language structures or standard library functions in the source code you develop in this window. LabWindows/CVI has code generation tools that streamline source code development. You will learn more about code generation tools in later chapters of this tutorial.

Editing Tools

In addition to standard Windows editing features—cut, copy, paste, and so on—the LabWindows/CVI Source window has a number of quick editing features that are helpful when you work with large source files or projects with a large number of source files. The arrow positioning keys, <Page Up>, <Page Down>, <Home>, and <End>, operate the same way as positioning keys in a word processor. Complete the following steps to view some of the editing features available in LabWindows/CVI.

1. Select **View»Line Numbers** to refer to particular line numbers. A new column to the left of the window shows line numbers.
2. Many times, the programs you develop in LabWindows/CVI refer to other files, such as header files or user interface files. To view these additional files quickly, place the cursor on the filename in the source code and select **File»Open Quoted Text**, press <Ctrl-U>, or right-click the filename and select **Open Quoted Text**.

Place the cursor on the `ansi_c.h` filename in `sample1.c` and press <Ctrl-U>.

LabWindows/CVI opens the `ansi_c.h` header file in a separate Source window. Scroll through the header file. Notice that it contains all of the standard header files defined for the standard ANSI C Library. Close the `ansi_c.h` header file.

3. If you want to view a portion of your source code while you make changes to another area of the source code in the same file, you can split the window into top and bottom halves called *subwindows*.

To split the window, click and drag the double line at the top of the Source window to the middle of the screen. Each subwindow contains a duplicate copy of the source code, as shown in Figure 2-3.

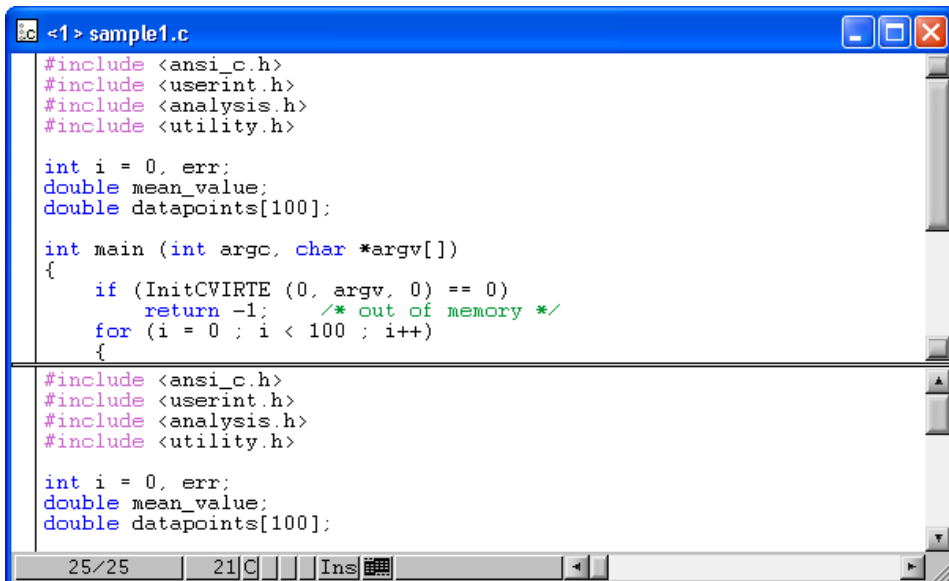


Figure 2-3. Split Source Window

Notice how each half of the window scrolls independently to display different areas of the same file simultaneously. Place the cursor on line 5 and type text. The text appears in both halves of the window.

4. If you make editing mistakes while entering or editing source code in the Source window, LabWindows/CVI has an **Undo** feature to reverse any mistakes. The default configuration of LabWindows/CVI allows up to 100 undo operations, and you can undo up to 1,000 operations. Select **Edit»Undo**. The text you entered on line 5 of the source code disappears.
5. Drag the dividing line between the two subwindows back to the top to make a single window again.
6. You can use two different methods to quickly move to a particular line of code in your source file. If you know the line number you want to view, select **View»Line** and enter the line number.

You also can set tags on particular lines to highlight lines of code to which you can jump quickly. Place the cursor on line 3. Select **View»Toggle Tag**. A green square appears in the left-hand column of the Source window.

Move the cursor to line 12 of the Source window and add another tag. Select **View»Next Tag**, and the cursor jumps to the next tagged line in your source code. You also can jump between tags by pressing the <F2> key.

7. LabWindows/CVI also displays the function prototype as you type. Move the cursor to a blank line in the code and type `FileSelectPopup` (. If you do not see the prototype after you type the parentheses, select **Edit»Show Prototype**.

In the function prototype, you can launch additional selection dialog boxes for parameters that provide them. In the `FileSelectPopup` prototype, click the `...` button next to the **DefaultDirectory** parameter to view the Open file dialog box.

You also can access function and parameter help from the function prototype. Place your cursor on the element you want to view help for and click the question mark or press `<F1>`.

8. You can use the **Edit»Show Completions** option to view a list of potential matches for functions or variables you are typing. On another blank line, type `Add` and press `<Ctrl-Space>` to view the drop-down list of matches.
9. Use `<Ctrl-Q>` to execute an incremental search. Move the cursor to the top of the file. Press `<Ctrl-Q>` and type `data`. Press `<Ctrl-Q>` two more times, and LabWindows/CVI highlights each instance of `data`.

Close `sample1.c` before moving on to the next section. You might be prompted to save changes from `sample1.prj`. Click **Discard** to continue.

Operating Projects with a User Interface

LabWindows/CVI makes text-based screen I/O very simple through the Standard Input/Output window. Most advanced applications, however, require you to build and operate a custom GUI to control the program flow and display the results. In Chapter 5, *Building a Graphical User Interface*, you learn how to build a GUI. This section introduces you to the way a GUI looks and works. Complete the following steps to run Sample 2.

1. Right-click **Sample2** in the Project Tree and select **Set Active Project**.
2. Run the project by selecting **Run»Debug sample2_dbg.exe** or by pressing `<Shift-F5>`. LabWindows/CVI launches the GUI shown in Figure 2-4 after the program compiles and runs.

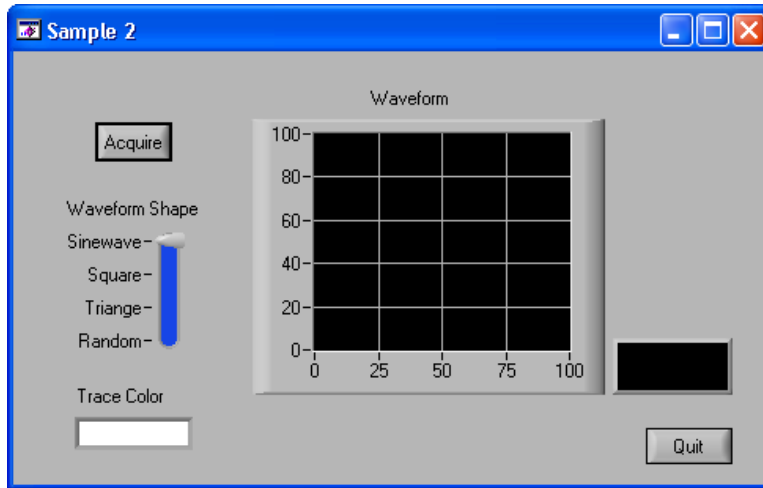


Figure 2-4. sample2.uir Panel when Running

3. Click **Acquire** to display a waveform on the graph control on the GUI. Experiment with the user interface controls, choosing different shapes and changing the color, then clicking **Acquire** to view your changes.
4. Click **Quit** to stop program execution.

In the remaining chapters of this tutorial, you will learn how to build a project similar to `sample2.prj`. You will learn about the tools for designing a GUI in LabWindows/CVI and about the code generation tools to develop the C source code for the project.

Interactive Code Generation Tools

In the first part of the tutorial, you learned how to load and run projects and edit source code in LabWindows/CVI. In this chapter, you will get acquainted with some of the tools available for interactive code generation in LabWindows/CVI.

Setting Up

Complete the following steps to prepare for this part of the tutorial.

1. Close all windows except the Workspace window.



Note If you have not saved the contents of the windows, LabWindows/CVI prompts you to do so. If you want to save the contents, click **Save** and enter a filename. If you do not want to save the contents, click **Discard**.

2. Make **Sample1** the active project.

Library Tree

All of the libraries in LabWindows/CVI appear under the Library Tree, as shown in Figure 3-1.



Note If you have the Full Development System, the Library Tree includes the Internet Library and the Advanced Analysis Library. If you have the Base package, the Library Tree includes the Analysis Library.

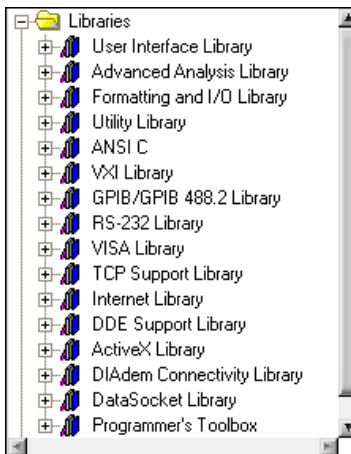


Figure 3-1. Library Tree

When you expand a library from the Library Tree, you can see the hierarchical structure of that library. Figure 3-2 shows the User Interface Library function tree.

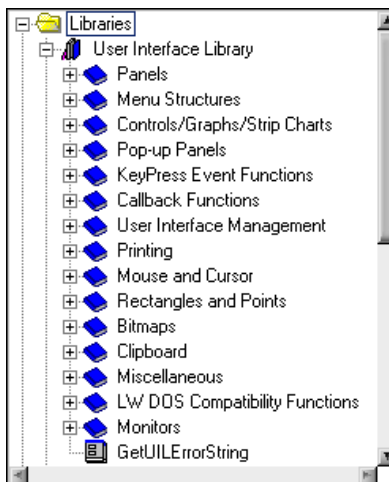


Figure 3-2. User Interface Library Function Tree

You can search quickly through the hierarchy of the library to find the right function. To find a particular function, right-click in the Library Tree, select **Find**, and enter the name of the function.

Accessing the User Interface Library

In this section of the tutorial, you will use the User Interface Library to display a graph of the random numbers generated in Sample 1. Complete the following steps:

1. Double-click `sample1.c` in the Project Tree to open the source code.
2. Verify that the program runs correctly by selecting **Run»Debug sample1_dbg.exe**. The project generates 100 numbers and displays them in the Standard Input/Output window.
3. Press <Enter> to close the Standard Input/Output window.
4. In `sample1.c`, place the cursor in the line above the `printf ("Press the <Enter> key to terminate....");` statement.
5. In the Library Tree, expand **User Interface Library»Pop-up Panels»Graph Popups»YGraphPopup** to open the window shown in Figure 3-3.

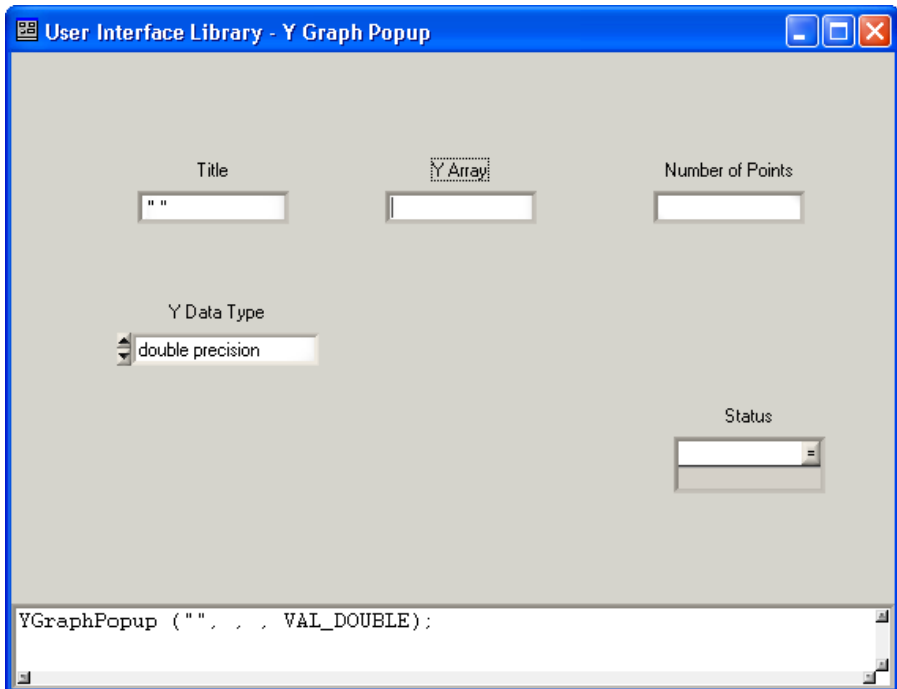


Figure 3-3. Y Graph Popup Function Panel

Function Panel Fundamentals

The window that LabWindows/CVI opens when you select **YGraphPopup** is called a function panel. A function panel is a graphical view of a library function in LabWindows/CVI. Function panels serve four important purposes in LabWindows/CVI.

- Function panels provide help that discusses the purpose of each function in the LabWindows/CVI libraries and of each parameter in the function call.
- You automatically can declare variables in memory to be used as function parameters from a function panel.
- With function panels, you can execute each LabWindows/CVI function interactively before incorporating it into the program. With this feature, you can experiment with the parameter values until you are satisfied with the operation of the function.
- Function panels generate code automatically, so that the function call syntax is inserted into your program source code.

Function Panel Controls

The controls on the function panel represent parameters. Enter values in the controls to specify parameter values. There are eight types of controls, and these controls are explained as you encounter them in the examples that follow. Some controls contain a ... button next to them. These controls provide additional dialog boxes to help you select input for parameters.

The text cursor is currently located in the **Title** control. The **Title** control is called an *input control*. You can enter a numeric value or variable name into an input control.

Function Panel Help

You can access help for functions and parameters from function panels. Table 3-1 lists methods for accessing the help information.

Table 3-1. Function Panel Help Display Procedures

Type of Help	How to View Help
Function Help	Select Help»Function or Help»Online Function Help . <i>or</i> Right-click anywhere on the background of the function panel.
Parameter Help	Place the cursor in the control, then select Help»Control . <i>or</i> Right-click the control. <i>or</i> Press <F1> from the control.
Combined Help	Select Help»Online Function Help . <i>or</i> Press <Ctrl-Shift-F1> to access the <i>LabWindows/CVI Help</i> .

Drawing a Graph

Use the Y Graph Popup function panel to create a line of code that graphs the array of random numbers that the sample program generates. Complete the following steps to generate the code.

1. Enter `Random Data` in the **Title** control. The title must remain within the quotation marks.
2. Enter the array name `datapoints` in the **Y Array** control.
3. Enter `50` in the **Number of Points** control.
4. The up and down arrows in the **Y Data Type** control indicate that it is a ring control. Press the up and down arrow keys on your keyboard to move through the ring of choices until you find **double precision**. Alternatively, you can click the ring control and select **double precision**.
5. Enter the variable name `err` in the **Status** control.

Confirm that your function panel matches the one shown in Figure 3-4.

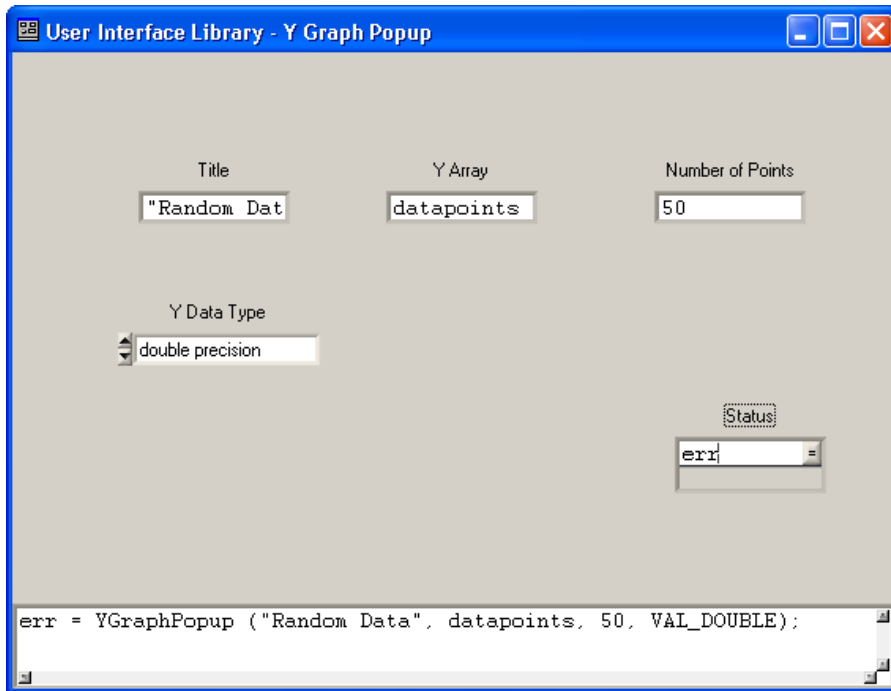
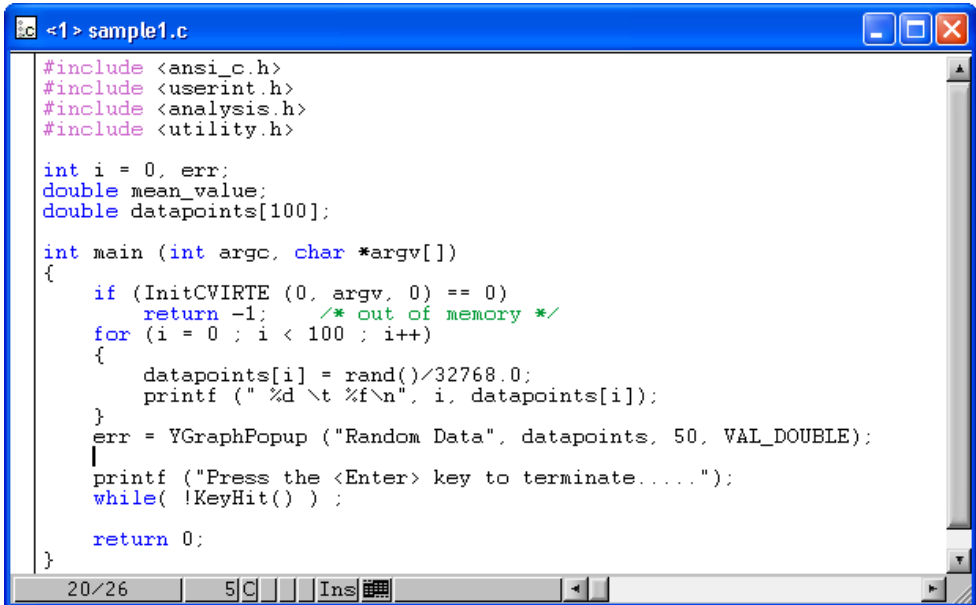


Figure 3-4. Completed Y Graph Popup Function Panel

Inserting Code from a Function Panel

The small window at the bottom of the function panel is the Generated Code pane. LabWindows/CVI updates the line of code in the Generated Code pane as you enter values into the controls on the function panel. To place these lines of code directly into your source code, complete the following steps:

1. Select **Code»Set Target File** and select **sample1.c** in the dialog box.
2. Select **Code»Insert Function Call**. LabWindows/CVI pastes the code from the Generated Code pane of the function panel to the `sample1.c` source code at the position of the text cursor in the source file, as shown in Figure 3-5. Confirm that the new function appears immediately after the For loop.



```

#include <ansi_c.h>
#include <userint.h>
#include <analysis.h>
#include <utility.h>

int i = 0, err;
double mean_value;
double datapoints[100];

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    for (i = 0 ; i < 100 ; i++)
    {
        datapoints[i] = rand()/32768.0;
        printf (" %d \t %f\n", i, datapoints[i]);
    }
    err = YGraphPopup ("Random Data", datapoints, 50, VAL_DOUBLE);
    printf ("Press the <Enter> key to terminate.....");
    while( !KeyHit() ) ;

    return 0;
}

```

Figure 3-5. Source Window with Code Inserted from Function Panel

3. Save the file.
4. To execute the program, select **Run»Debug sample1_dbg.exe**. As the program code executes, the Standard Input/Output window displays the screen output, then your program displays the graph of the data.
5. Press <Enter> or click **OK** to close the graph window and return to the Standard Input/Output window. Then press <Enter> to close the Standard Input/Output window and return to the Source window.

Analyzing Data

Now use a function from the LabWindows/CVI Advanced Analysis (or Analysis) Library to calculate the mean of the values in the array. Before continuing, position the cursor in the Source window on the line beneath the following statement:

```
err = YGraphPopup ("Random Data", datapoints, 50, VAL_DOUBLE);
```

Complete the following steps to generate a call to the `Mean` function and insert it into the source code.

1. Right-click the Library Tree and select **Find**.
2. In the Find dialog box, enter `Mean` and click **Find Next**.
3. **Mean** is highlighted. Double-click **Mean** to open the function panel.

4. Enter the array name `datapoints` in the **Input Array** control.
5. Enter `100` in the **Number of Elements** control.
6. Leave the remaining controls empty and go to the next section.

Output Values on a Function Panel

The **Mean** control on the Mean function panel is an output control. An output control displays data that results from executing a function. You can enter your own variable name in which to store the results. Enter a variable name if you intend to generate code for your program. For this example, enter a variable name as follows:

1. Enter `&mean_value` in the **Mean** control. Confirm that the function panel matches the one in Figure 3-6.

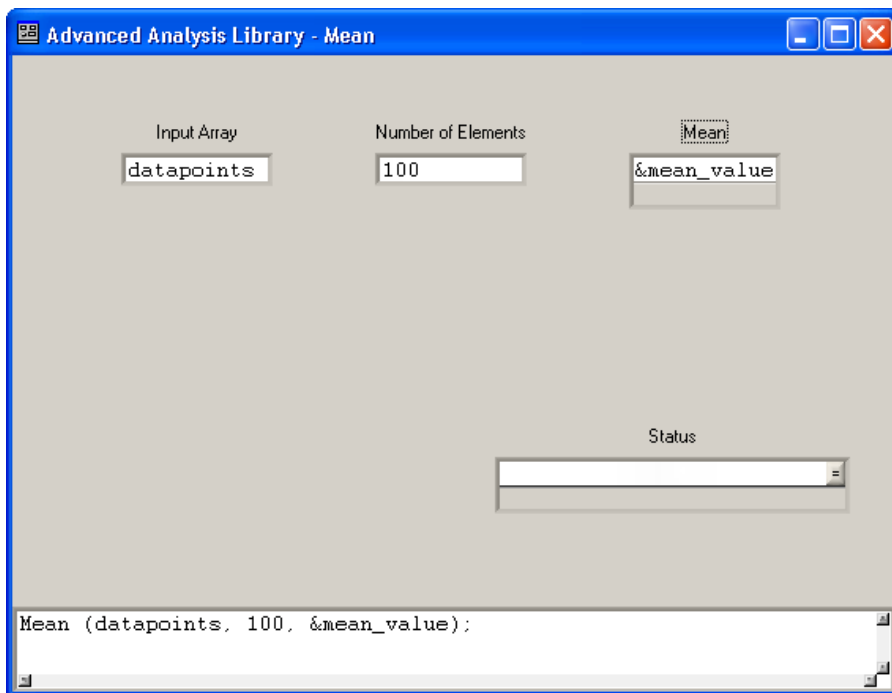


Figure 3-6. Completed Function Panel for Mean

2. Select **Code»Insert Function Call**, or press `<Ctrl-I>`, to insert the line of code for calling the Mean function into `sample1.c`.
3. LabWindows/CVI inserts the code on the line where you left the cursor in the Source window.

Recalling a Function Panel

Notice that the function call to `YGraphPopup` graphs only 50 elements of the `datapoints` array. To change this line of code to graph all 100 elements of the array, you can either modify the code directly in the Source window, or you can modify the function panel associated with the `YGraphPopup` function. Complete the following steps to edit this line of code using the Recall Function Panel feature.

1. Place the cursor in the Source window anywhere in the following function call:

```
err = YGraphPopup ("Random Data", datapoints, 50, VAL_DOUBLE);
```
2. Select **View»Recall Function Panel**. The Y Graph Popup function panel appears. Notice that the controls automatically reflect the state of the function call in the Source window.
3. Change **Number of Points** to 100.
4. Insert the new code into your source code by selecting **Code»Insert Function Call**. In the dialog box, you can either replace the current function call or insert a new line with the new function panel values below the existing call.
5. Click **Replace** and return to the Source window. Notice that in `sample1.c` the call to `YGraphPopup` is now set to graph 100 elements of the `datapoints` array.

Finishing the Program

Now you have a program that generates 100 random numbers, plots the numbers on a graph, and calculates the mean value.

1. As a final step, type the following line above the `printf ("Press the <Enter> key to terminate....");` statement in the Source window.

```
printf ("Mean = \t %f \n", mean_value);
```

2. Confirm that your completed source code matches the following code:

```
#include <ansi_c.h>
#include <userint.h>
#include <analysis.h>
#include <utility.h>

int i= 0, err;
double mean_value;
double datapoints[100];

int main(int argc, char *argv[])
{
    if (InitCVRTE (0, argv, 0) == 0)
        return -1;    /* out of memory */

    for (i = 0 ; i < 100 ; i++)
```

```

    {
        datapoints[i] = rand()/32768.0;
        printf (" %d \t %f\n", i, datapoints[i]);
    }
    err = YGraphPopup ("Random Data", datapoints, 100, VAL_DOUBLE);
    Mean (datapoints, 100, &mean_value);
    printf ("Mean = \t %f \n", mean_value);
    printf ("Press the <Enter> key to terminate.....");
    while ( !KeyHit() ) ;

    return 0;
}

```

3. Save the source file and the project, if needed.
4. Select **Run»Debug sample1_dbg.exe** to execute the program. The program first prints out the random numbers in the Standard Input/Output window as they are calculated. Next, it draws a plot of the data. After you press **OK** in the Random Data panel, the program calculates the mean of the numbers and prints it in the Standard Input/Output window after the last output line.
5. Close the Source window before going on to the next part of the tutorial.

Interactively Executing a Function Panel

In this chapter, you learned how to use function panels to interactively build function calls into programs. You also learned that function panels can teach you how functions operate through the help. Function panels are powerful because they permit you to execute functions interactively, before you insert the function call into your source code. In Chapter 6, *Using Function Panels and Libraries*, and Chapter 7, *Adding Analysis to Your Program*, you will learn how to use function panels to declare variables for use in your program and how to run functions interactively.

Executing and Debugging Tools

In this chapter, you will become acquainted with the following tools available for executing and debugging in the interactive LabWindows/CVI environment.

- Step modes of execution
- Breakpoints
- Variables window
- Array Display
- String Display
- Watch window
- Graphical Array View

Setting Up

1. Make **Sample3** the active project.
2. Open `sample3.c`.

The `sample3.c` program uses the same random number function as the `sample1.c` program that you ran in Chapter 2, *Loading, Running, and Editing Source Code*.

Step Mode Execution








Step mode execution is a useful run-time tool for debugging programs. To step through `sample3.c`, complete the following steps:

1. Select **Run»Break on»First Statement** to stop execution at the first statement in the source code.
2. Select **Run»Debug sample3_dbg.exe** to begin execution of the program. After the program compiles, the `main` function line in the program is highlighted in the Source window, indicating that program execution is currently suspended.
3. To execute the highlighted line, select **Run»Step Into**.



Tip To avoid accessing the **Run** menu each time you perform step mode execution, use the shortcut key combinations listed in Table 4-1. You also can click the icons in the toolbar to execute these commands.

Table 4-1. Quick Keys for Step Mode Execution

Command	Shortcut Key Combination	Toolbar Icon	Description
Continue	<F5>		Causes the program to continue operation until it completes or reaches a breakpoint
Go to Cursor	<F7>		Continues program execution until the program reaches the location of the cursor
Set Next Statement	<Ctrl-Shift-F7>		Changes the next statement to execute
Step Into	<F8>		Single-steps through the code of the function call being executed
Step Over	<F10>		Executes a function call without single-stepping through the function code itself
Finish Function	<Ctrl-F10>		Resumes execution through the end of the current function and breakpoints on the next statement
Terminate Execution	<Ctrl-F12>		Halts execution of the program during step mode

- To find the definition of the `get_and_print_random` function, click the call to the function on line 17 of `sample3.c` and select **Edit»Go to Definition**. Alternatively, you can right-click the call function and select **Go to Definition**.

The **Go to Definition** command immediately finds the definition of the function, even when the function resides in a different source file. However, the target source file must have been compiled in the project. You also can use this command to find variable declarations.

- Use **Step Into** to begin stepping through the program. Notice that when the `get_and_print_random` function is executed, the highlighting moves to the function and traces the instructions inside the function. Continue to step through the program until you have created several random values.
- You can select the next statement to execute with the **Run»Set Next Statement** command. Step through the program until you are in the `main` function. Place the cursor on the line with the call to `printf ("Press any key to terminate")`. Select **Run»Set Next Statement**. The highlighting moves to that line. Press <F8>. Notice that the Standard I/O window contains the output of the `printf` statement.

Breakpoints

Breakpoints are another run-time tool that you can use to debug programs in LabWindows/CVI. A breakpoint is a location in a program at which LabWindows/CVI suspends execution of your program. You can invoke a breakpoint in LabWindows/CVI in the following six ways:

- **Predetermined Breakpoint**—Insert a Breakpoint icon in the Source window.
- **Instant Breakpoint**—Press <Ctrl-F12> while a window is active in the LabWindows/CVI environment.
- **Breakpoint on Library Errors**—Cause LabWindows/CVI to pause when a library function returns an error.
- **Conditional Breakpoint**—Cause LabWindows/CVI to pause when a user-specified condition becomes true.
- **Programmatic Breakpoint**—In your code, call the `Breakpoint` function.
- **Watch Expression Breakpoint**—Cause LabWindows/CVI to pause when the value of a watch expression changes.

The following sections explain predetermined breakpoints and instant breakpoints. For more information about conditional breakpoints and the **Break on»Library Errors** feature, refer to *Break on Library Errors* in the **Index** of the *LabWindows/CVI Help*.

Predetermined Breakpoints

To insert a breakpoint at a specific location in your source code, click in the left column of the Source window on the line you want to break on. Complete the following steps to insert a breakpoint inside the For loop so the program halts after it returns from the function call.

1. Stop program execution by selecting **Run»Terminate Execution**.
2. Disable **Run»Break on»First Statement**. The checkmark next to the **First Statement** menu item disappears.
3. In the Source window, click to the left of the line that contains the following statement:

```
get_and_print_random (i, &my_array[i]);
```

A red diamond, which represents a breakpoint, appears beside that line as shown in Figure 4-1.



Note You do not need to suspend or terminate execution to insert a breakpoint. If you insert a breakpoint while the program is running, LabWindows/CVI suspends the program when it reaches that line of code.

```

1  #include <ansi_c.h>
2  #include <utility.h>
3
4  void get_and_print_random (int, double *);
5
6  int main (int argc, char *argv[])
7  {
8      double my_array[100];
9      int i;
10
11     if (InitCVIRTE (0, argv, 0) == 0)
12         return -1; /* out of memory */
13
14     Cls ();
15     for (i = 0; i < 100; i++)
16     {
17         get_and_print_random (i, &my_array[i]);
18     }
19
20     printf ("Press any key to terminate.....");
21     while( !KeyHit() );
22
23     return 0;
24 }
25
26 void get_and_print_random (int index, double *random_val)
27 {
28     *random_val = (double)rand () / 32767.0;
29     printf ("%d \t %f \n", index, *random_val);
30 }

```

Figure 4-1. Breakpoint beside a Line of Code

4. Begin execution of the program by selecting **Run»Debug sample3_dbg.exe**. When LabWindows/CVI encounters the breakpoint during execution, it suspends program execution and highlights the line where you inserted the breakpoint.
5. Press <F5> to continue execution. Program execution continues until the next breakpoint or until completion. You can single-step through the code at that point by selecting **Run»Step Over** or **Run»Step Into**.
6. Stop the program at a breakpoint by pressing <Ctrl-F12> or by selecting **Run»Terminate Execution**.
7. To remove the breakpoint from the program, click the red diamond.

Instant Breakpoints

Complete the following steps to enter a breakpoint using <Ctrl-F12> after program execution has begun.

1. Click any LabWindows/CVI environment window to make it active.
2. Select **Run»Debug sample3_dbg.exe** to begin program execution.
3. When the program begins running, press <Ctrl-F12>.

The program enters breakpoint mode just as it did with the breakpoint symbol. However, in this case, the next executable statement in the program appears highlighted, not as a line with a breakpoint symbol. At this point, you can use all LabWindows/CVI options for continuing execution or stepping through execution.

4. Press <Ctrl-F12> again or select **Run»Terminate Execution** to stop the program.

Displaying and Editing Data

Step mode execution and breakpoints are useful tools for high-level testing. However, many times you need to look beyond your source code to test your programs. The LabWindows/CVI interactive environment provides the following displays for viewing and editing the data for your program.

- Variables window
- Array Display window
- String Display window
- Watch window
- Tooltips
- Graphical Array View

Variables Window

The Variables window shows all variables currently declared in the LabWindows/CVI interactive program. To view the Variables window, select **Window»Variables**.

The Variables window lists the name, value, and type of currently active variables. LabWindows/CVI displays variables in categories according to how they are defined, such as global or local. The Stack Trace section shows the current call stack of functions. To view variables that are active elsewhere in the call stack, double-click the corresponding function in the Stack Trace.

You can view the Variables window at any time to inspect variable values. This feature is especially useful when you step through a program during execution that has stopped at a breakpoint. Complete the following steps to step through the program and view the Variables window at different points in the execution of the program.

1. Select **Run»Break on»First Statement**.
2. Select **Run»Debug sample3_dbg.exe**, or press <Shift-F5>, to run the program. When the program begins execution, LabWindows/CVI highlights the `main` function in the Source window.
3. Select **Window»Variables** to view the Variables window, shown in Figure 4-2.

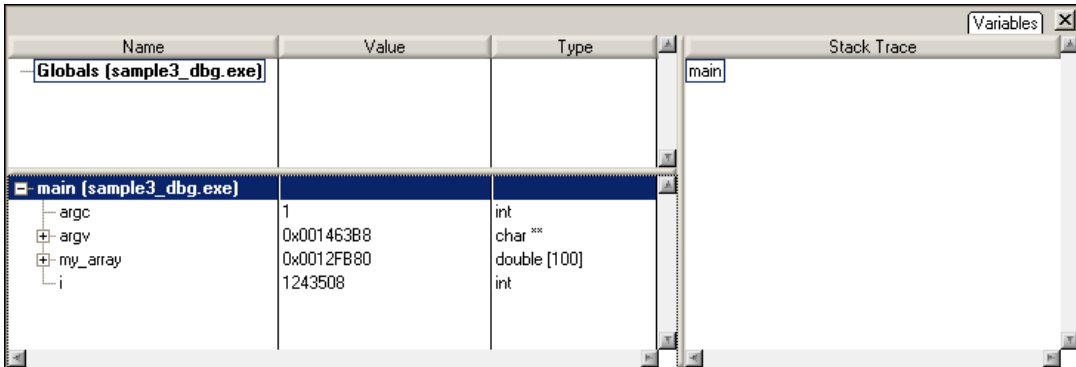


Figure 4-2. Variables Window during Execution of main

Notice that local variables appear under *main*: a double-precision array (*my_array*) and an integer (*i*). The Stack Trace section lists the current call stack of functions. To change the set of local variables, select a different function in the Stack Trace section.



Note The values you see for your project might differ from the values shown in Figure 4-2.

4. Select **Run»Step Into** until LabWindows/CVI highlights the line `*random_val = (double)rand () / 32767.0;`, which is the first statement in the `get_and_print_random` function.
5. In the Variables window, LabWindows/CVI now lists `get_and_print_random` in the Stack Trace section. The Variables window shows the variables that are declared locally to that function.
6. Leave the program in breakpoint mode and continue with the next section, *Editing Variables*.

Editing Variables

In addition to displaying variables, you can use the Variables window to edit the contents of a variable. The following steps describe how to use the Variables window for this purpose.

1. Make sure the `sample3.c` program is still in breakpoint mode on the following line inside the `get_and_print_random` function.


```
*random_val = (double)rand () / 32767.0;
```
2. Select **Run»Step Into** until the For loop executes a few times and the highlighting appears on the following statement:


```
get_and_print_random (i, &my_array[i]);
```

3. Highlight the `i` variable and select **Run»View Variable Value**. LabWindows/CVI highlights the `i` variable in the Variables window.
4. From the Variables window, press <Enter> to edit the value of `i`. Enter 10 in the value column and press <Enter>.
5. In the Source window, select **Run»Step Into**.
6. Position the Standard Input/Output window so you can see it and the Workspace window. Step through the source code until the next random number appears in the Standard Input/Output Window.

Notice that the index is now 10. The change you made using the Variables window took effect immediately in the execution of the program.

Array Display

Another useful data display in the LabWindows/CVI interactive program is the Array Display. The Array Display shows the contents of an array of data. You can use the Array Display to edit array elements in the same way that you edited variables using the Variables window.

1. Confirm that the program is still in breakpoint mode.
2. Select **Run»Step Into** or press <F8> until the highlighting reaches the following line:


```
printf (" %d \t %f \n", index, *random_val);
```
3. Double-click `main` in the Stack Trace to view the variables defined in `main`. In the Variables window, double-click `my_array` to view `my_array` in the Array Display, as shown in Figure 4-3.

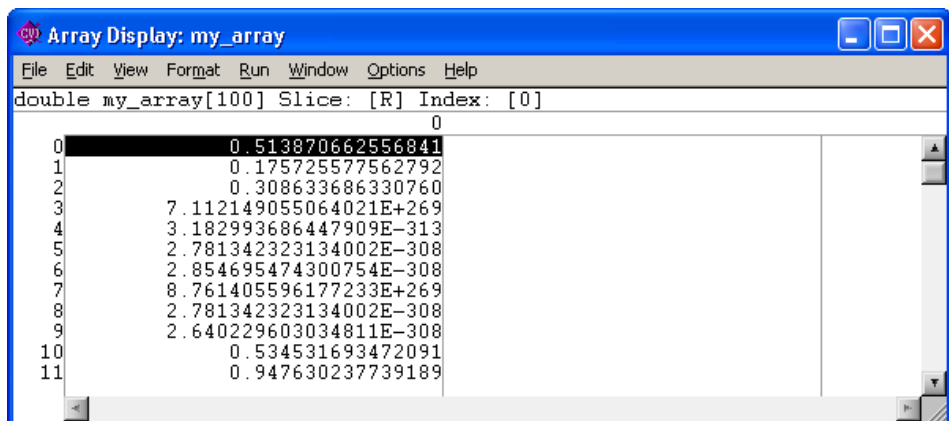


Figure 4-3. Array Display



Note The actual values in your array might differ from the values shown in Figure 4-3. This example generates numbers between 0 and 1. The numbers shown in Figure 4-3 for index 3 through 9 are invalid, uninitialized values.

The Array Display shows the values of array elements in tabular format. In Figure 4-3, the array, `my_array`, is a one-dimensional array, so the display consists of one column of numbers. The numbers in the column on the left side of the display indicate the index number. The first element is zero.

Take a moment to scroll through the display.

Editing Arrays

You can edit individual elements in the array just as you edited variables in the Variables window. For example, to edit element 12 of the array, complete the following steps:

1. Double-click or highlight element 12 (index 11) in the array and press <Enter> to open the Edit Value dialog box.
2. Enter the value 0.5 and press <Enter>. Notice that element 12 is now set to 0.5.
3. In the Source window, press <F8>, or select **Run»Step Into**, to execute the `printf` statement. Notice that the random value printed in the Standard Input/Output window is the value that you entered, 0.5.
4. Press <F5>, or select **Run»Continue** to complete program execution.

String Display

Another useful data display is the String Display. Select a string variable from the Variables window to launch the String Display. The String Display is similar to the Array Display except that you use the String Display to view and edit elements of a string. Operations in the String Display are similar to the operations you performed in the Array Display. For more information about the String Display, refer to *Using LabWindows/CVI»Debugging Tools»Using the Array and String Display Windows* in the *LabWindows/CVI Help*.

Watch Window

The Watch window is a powerful debugging tool because you can view values of variables changing dynamically as your program executes. You also can use the Watch window to view expression values and set conditional breakpoints when variable or expression values change. The following steps show you how to use the Watch window to view variables during program execution.

1. With `sample3.prj` still loaded as the current project, make sure that **Run»Break on»First Statement** is still enabled.
2. Select **Run»Debug sample3_dbg.exe**, or press <Shift-F5>, to start program execution. Execution breaks with the `main` function highlighted.

- Press <F8> until the highlighting reaches the following line:

```
*random_val = (double)rand() / 32767.0;
```
- In the Variables window, highlight the `random_val` variable.
- Select **Options»Add Watch Expression** to indicate that you want to assign the `random_val` variable to the Watch window. The dialog box shown in Figure 4-4 appears.

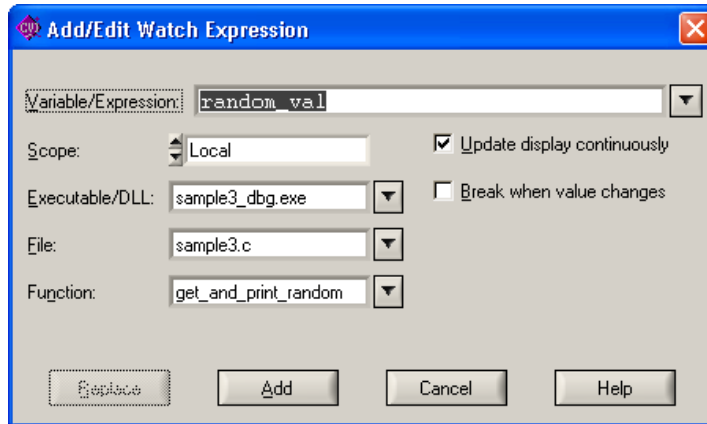


Figure 4-4. Add/Edit Watch Expression Dialog Box

- Make sure that the **Update display continuously** option is enabled and click **Add**.
- Select **Run»Continue** to complete program execution. As the program continues running, you can watch as the value of the `random_val` variable changes dynamically in the Watch window.

Tooltips

You also can use the following method to edit variables:

- Make sure the **Break on»First Statement** option is still enabled and select **Run»Debug sample3_dbg.exe**.
- Step through the code until you reach the following line: `get_and_print_random(i, &my_array[i]);`.
- Position the mouse cursor on the first `i` variable in the `get_and_print_random` statement.
- The variable value appears in a tooltip. Highlight the 0 and enter 3.
- Step through the code to generate another random number.
- In the Standard Input/Output window, the index is 3.
- Select **Run»Continue** to complete program execution.

Graphical Array View

The Graphical Array View shows the values of arrays in a graph view. This display is available for 1D and 2D arrays during debugging. To open the Graphical Array View, complete the following steps:

1. Make sure the **Break on»First Statement** option is still enabled and select **Run»Debug sample3_dbg.exe**.
2. Step through the code until you reach the following line:

```
void get_and_print_random (int index, double *random_val).
```
3. In the Variables window, highlight the `random_val` variable and select **View»Graphical Array View** to view the `random_val` values in a graph. You also can right-click the variable name in the Source window and select **Graphical Array View**.
4. Select **Run»Continue** to complete program execution.

Building an Application in LabWindows/CVI

- Chapter 5, *Building a Graphical User Interface*, contains instructions for building a project consisting of a GUI and a C source file.
- Chapter 6, *Using Function Panels and Libraries*, contains instructions for using LabWindows/CVI function panels to generate code. You then will use this code to plot the graph control array on the user interface that you built in Chapter 5, *Building a Graphical User Interface*.
- Chapter 7, *Adding Analysis to Your Program*, contains instructions for adding analysis capabilities to your program to compute the maximum and minimum values of the random array you generate. To do this, you will write a callback function that finds the maximum and minimum values of the array and displays them in numeric readouts on the user interface.
- Chapter 8, *Using an Instrument Driver*, contains instructions for using an instrument driver. An instrument driver is a set of functions used to program an instrument or a group of related instruments. The high-level functions in an instrument driver incorporate many low-level operations including GPIB, VXI, or RS-232 read and write operations, data conversion, and scaling. The sample module in this chapter does not communicate with a real instrument but illustrates how an instrument driver is used in conjunction with the other LabWindows/CVI libraries to create programs.
- Chapter 9, *Additional Exercises*, contains exercises to help you learn more about the concepts you used throughout this tutorial. Each exercise builds on the code that you developed in the previous exercise.

Building a Graphical User Interface

In the first part of this tutorial, you executed a sample program that was controlled with a GUI developed in the User Interface Editor. In the remaining chapters of this tutorial, you will develop a project that consists of a GUI controlled by a C source file. In this chapter, you will learn to design a user interface with the User Interface Editor.

You can use the User Interface Editor to create a GUI for an application. A user interface contains objects such as menu bars, controls, and pop-up menus, all of which reside on a panel. In Chapter 6, *Using Function Panels and Libraries*, you will learn about the User Interface Library, which includes a set of functions to control the interface programmatically.

User Interface Editor

The User Interface Editor is an interactive drag-and-drop editor for designing custom GUIs. You can select a number of different controls from the **Create** menu and position them on the panels you create. You can customize each control through a series of dialog boxes in which you set attributes for the control appearance, settings, hot key connections, and label appearance.

Source Code Connection

After you design a user interface in the User Interface Editor, you can write C source code to control the GUI. To connect elements on the `.uir` to the source code, you must assign a name to each panel, menu, and control on your user interface. Then, you can use those names in the C source code to differentiate the controls on the GUI. You also can assign a function name to a control that is called automatically when you operate that control during program execution. Associate a constant name and a callback function with a control in the User Interface Editor within the Edit dialog box for the control.

After you complete a user interface and save it as a user interface resource (`.uir`) file, LabWindows/CVI automatically generates an include file that defines all the constants and callback functions you have assigned.

CodeBuilder

After you complete the `.uir` file, you can use the CodeBuilder utility in LabWindows/CVI to create a complete source file. CodeBuilder automatically includes in the source file the callback functions specified in your `.uir` file.

Sample Project

In the next few chapters of this tutorial, you will build a sample program that acquires and displays a waveform on a GUI. The development process includes the following steps:

1. Create a user interface in the User Interface Editor (this chapter).
2. Generate a shell program source file using CodeBuilder (this chapter).
3. Add to the C source code to generate and display the waveform (Chapter 6, [Using Function Panels and Libraries](#)).
4. Develop a callback function to compute the mean value of the waveform (Chapter 7, [Adding Analysis to Your Program](#)).
5. Use an instrument driver in the project to simulate data acquisition (Chapter 8, [Using an Instrument Driver](#)).

Setting Up

Close all windows except the Workspace window. In this chapter, you will create a new project and add it to the existing tutorial workspace.

Building a User Interface Resource (.uir) File

Complete the following steps to create the user interface of the sample project, as shown in Figure 5-1.



Tip Use the commands in the **Edit** menu and the **Arrange** menu to cut, copy, paste, align, and space user interface controls in the editor. Use the grid lines on the panel to align controls.

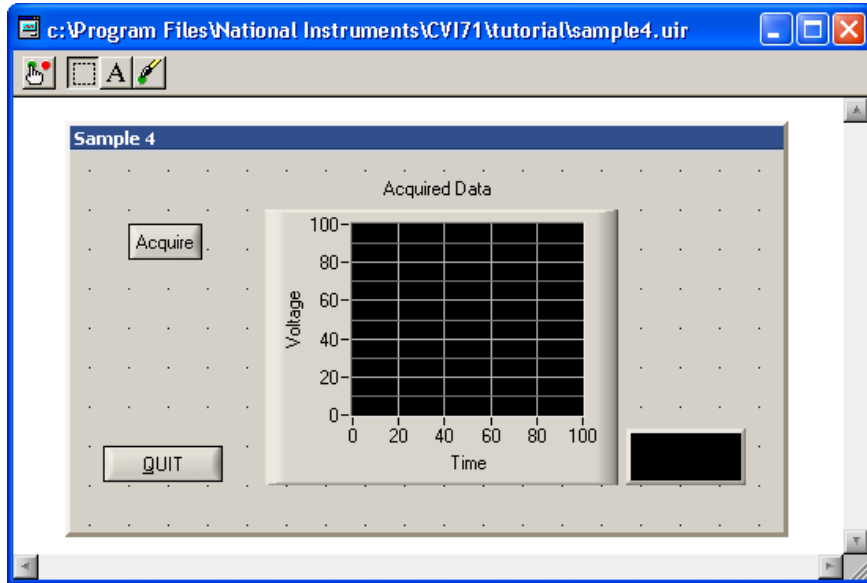


Figure 5-1. sample4.uir

Creating a .uir File

1. Select **File»New»User Interface (*.uir)** to create a new user interface resource file with a blank panel.
2. Double-click the panel to open the Edit Panel dialog box. In the Edit Panel dialog box, enter `Sample 4` as the **Panel Title**.
3. Click **OK** to close the dialog box.

Adding Command Buttons

1. Select **Create»Command Button»Square Command Button**. LabWindows/CVI places a button labeled **OK** on the panel.
2. To edit the button attributes, double-click the button or press `<Enter>`.
3. Assign a constant name to the button. The C source code uses this constant name to communicate with the button. LabWindows/CVI creates a default name for you, but you can assign your own constant names to your `.uir` file. Type `ACQUIRE` as the **Constant Name**.
4. Assign a function name that the program will call when a user clicks the **Acquire** button. Type `AcquireData` in **Callback Function**. In Chapter 6, *Using Function Panels and Libraries*, you will write the source code for the `AcquireData` function.

5. To change the label on the command button, type `Acquire` in place of the existing characters, `__OK`, in **Label**. If you type a double underscore before any letter in **Label**, the letter is underlined in the label. The user can select the control by pressing `<Alt>` and the underlined letter, provided that no accessible menu bars contain a menu with the same underlined letter.
6. (Optional) Click **Label Style** to customize the font for your button. Click **OK** in the Label Style dialog box when you finish.
7. Click **OK** in the Edit Command Button dialog box.
8. To add the **QUIT** button, select **Create»Custom Controls»Quit Button**. Custom controls are frequently used control configurations. The **QUIT** Button already has a callback function, `QuitCallback`, assigned.

Adding a Graph Control to the User Interface

1. Select **Create»Graph»Graph**. LabWindows/CVI places a graph control named Untitled Control on the panel.
2. To size the graph, click and drag one of the corners.
3. Double-click the graph control to open the Edit Graph dialog box in which you customize the graph attributes.
4. Type `WAVEFORM` in the **Constant Name** control.



Note Because the graph serves only as an indicator to display a waveform, the graph does not require a callback function. Callback functions are necessary only when the operation of the control initiates an action or acts as an input control. Indicators generally do not require callback functions.

5. Type `Acquired Data` in the **Label** control.
6. Use the **Bottom X-axis** and **Left Y-axis** buttons to assign `Time` and `Voltage` labels to the x- and y-axes, respectively. Confirm that the dialog box looks as shown in Figure 5-2.

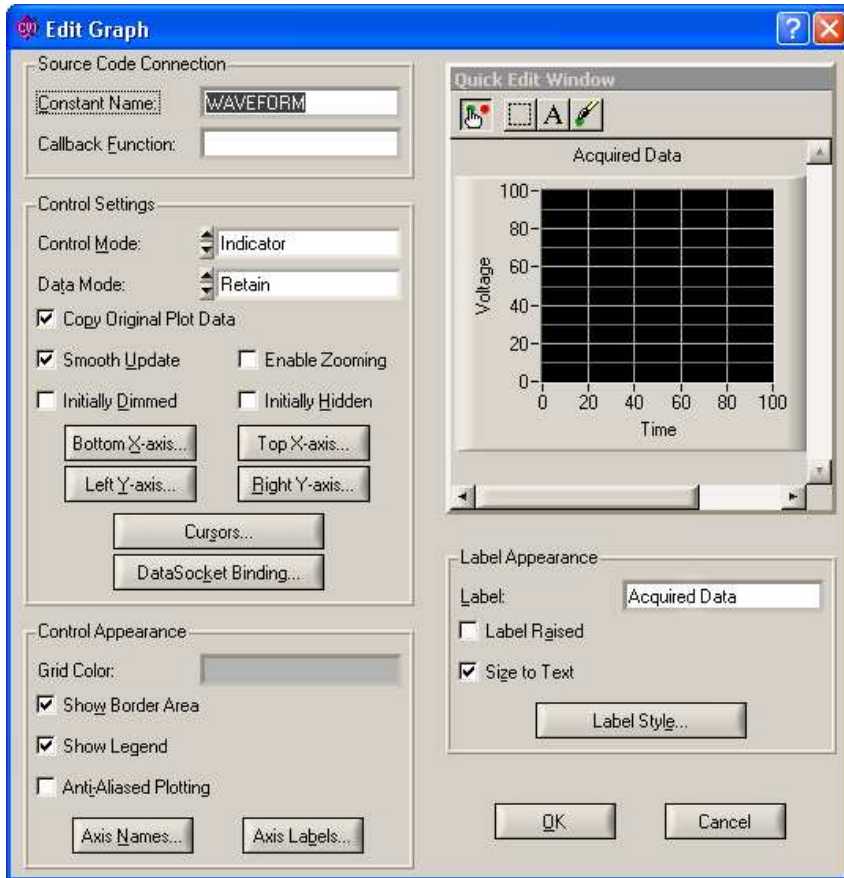


Figure 5-2. Edit Graph Dialog Box

7. After you set the graph attributes, click **OK** in the Edit Graph dialog box to close the dialog box.

Verify that the completed user interface looks like the one shown in Figure 5-1.

Saving the .uir File

1. Save the .uir file with the new controls as `sample4.uir`.
2. Select **View»Preview User Interface Header File** to inspect the header file that LabWindows/CVI has automatically created.
3. Close the header file. You do not need to save it.

Generating the Program Shell with CodeBuilder

Now that you have built a GUI in the User Interface Editor, the CodeBuilder feature can automatically generate a program shell for your GUI.

1. Before you use CodeBuilder, you must specify the events to which your program must respond. Select **Code»Preferences»Default Control Events**. In the Control Callback Events dialog box, shown in Figure 5-3, there is a checkmark beside the `EVENT_COMMIT` callback event and a checkmark beside the `EVENT_TIMER_TICK` callback event.

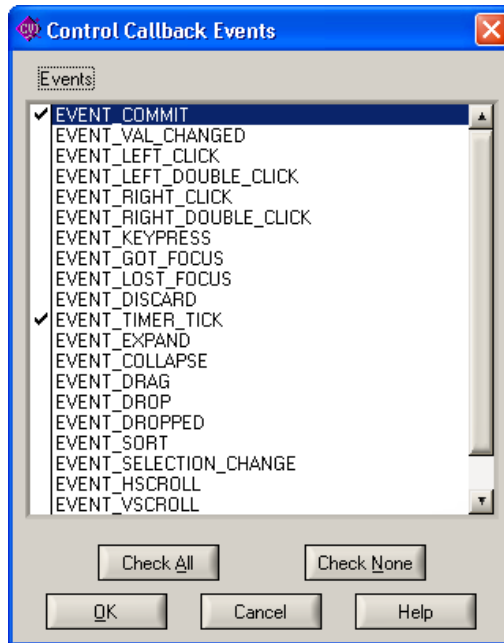


Figure 5-3. Control Callback Events Dialog Box

2. Later in this tutorial, you will develop code to display help when a user right-clicks a GUI control. To establish this functionality, you now must place a checkmark beside `EVENT_RIGHT_CLICK` in the list. Click **OK** to save your settings.

Now, your program can respond to the following two events:

- `EVENT_COMMIT`—A commit event (click or <Enter>) that generates data and plots it on the graph
 - `EVENT_RIGHT_CLICK`—A right-click event that displays help
3. Select **Code»Generate»All Code** to open the Generate All Code dialog box. You need to specify several options in this dialog box.

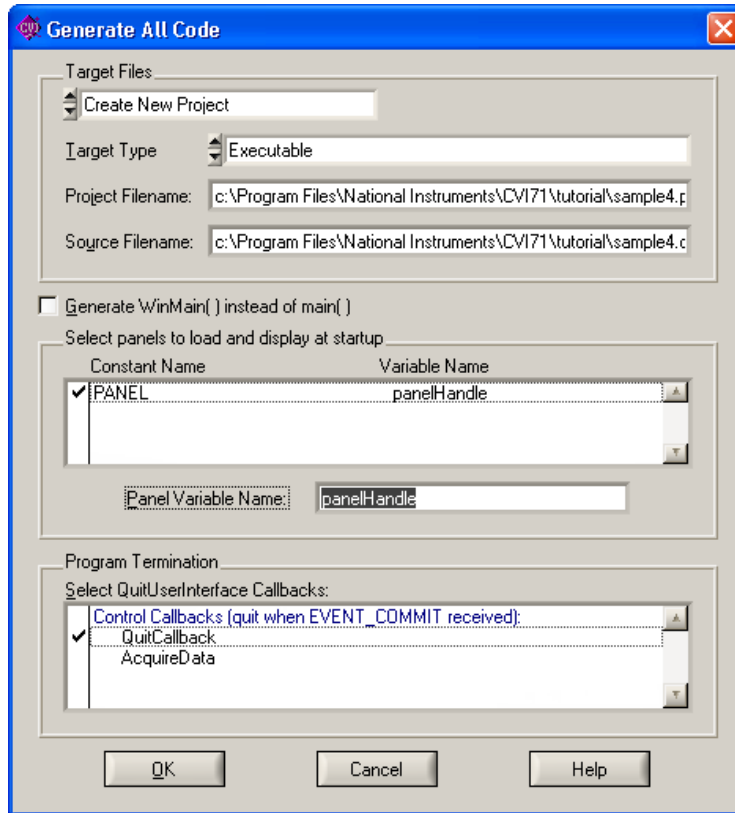


Figure 5-4. Generate All Code Dialog Box

4. In the Target Files section, select **Create New Project** and select **Executable** as the **Target Type**.
5. You must decide which panels to display at program startup. In this case, only one user interface panel exists. Select this item so that this panel displays at program startup.



Note For this exercise, the panel variable name must be `panelHandle`. Type the correct name in **Panel Variable Name**, if necessary.

6. The Program Termination section lists the callback functions in the `.uir` file. Select a function from this list that causes the program to terminate execution. In this case, select the **QuitCallback** function in the dialog box so that a checkmark appears next to it. Verify that the Generate All Code dialog box matches the one shown in Figure 5-4.
7. Click **OK**.
8. Click **Yes** when prompted to unload the current project.

9. In the New Project Options dialog box, enable the **Create Project in Current Workspace** option and click **OK**.
10. CodeBuilder builds the source code for `sample4.prj`. LabWindows/CVI adds the project to your workspace and creates a new Source window with the following code:

```
#include <cvirte.h>
#include <userint.h>
#include "sample4.h"

static int panelHandle;

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return (-1); /* out of memory */
    if ((panelHandle = LoadPanel (0, "sample4.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);
    return 0;
}

int CVICALLBACK AcquireData (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:

            break;

        case EVENT_RIGHT_CLICK:

            break;

    }
    return 0;
}

int CVICALLBACK QuitCallback (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
    }
}
```

```
        case EVENT_RIGHT_CLICK:  
            break;  
        }  
        return 0;  
    }
```

Using Function Panels and Libraries

In this chapter of the tutorial, you will use LabWindows/CVI function panels to generate code. You will then use this code to plot an array on the graph control on the user interface that you built in Chapter 5, *Building a Graphical User Interface*.

Setting Up

If you did not proceed directly from Chapter 5, *Building a Graphical User Interface*, go back and do so now. Disable the **Run»Break on»First Statement** option.

Analyzing the Source Code

The source code that you generated for the Sample 4 program is incomplete. In this chapter, you will add code to the program to complete it. The program consists of three functions. It is important that you understand what tasks each function in the `sample4.c` code performs because you will write similar functions in the future for your own LabWindows/CVI programs.

main Function

The `main` function is simple and represents the first step you need to take when you build your own applications. The `main` function is shown in the following code:

```
int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    if ((panelHandle = LoadPanel (0, "sample4.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);
    return 0;
}
```

Before you can display or operate the user interface that you created, you must first load the panel from the `.uir` file on your hard disk into memory.

- The `LoadPanel` function performs this operation in the `main` function.
- The `DisplayPanel` function displays the panel on the screen.
- The `RunUserInterface` function activates LabWindows/CVI to begin sending events from the user interface to the C program you are developing.
- The `DiscardPanel` function removes the panel from memory and from the screen.

AcquireData Function

The `AcquireData` function automatically executes whenever you click **Acquire** on the user interface. You will add to this function later in this chapter so you can plot the array on the graph control that you created on the user interface. The `AcquireData` function is shown in the following code:

```
int CVICALLBACK AcquireData (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}
```

QuitCallback Function

The `QuitCallback` function automatically executes whenever you click **QUIT** on the user interface. This function disables the user interface from sending event information to the callback function and causes the `RunUserInterface` call in the `main` function to return. The `QuitCallback` function is shown in the following code:

```
int CVICALLBACK QuitCallback (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
        case EVENT_RIGHT_CLICK:
```

```

        break;
    }
    return 0;
}

```

Generating a Random Array of Data

Complete the source code for `sample4.c` so the program generates an array of random numbers and plots the array on the graph control. Most of the action takes place in the `AcquireData` function. When a user clicks **Acquire**, the program generates a random number using the ANSI C `srand` and `rand` functions.

1. On the line following `case EVENT_COMMIT` in the `AcquireData` function, enter the following lines of code to generate the random numbers.

```

srand (time(NULL));
amp = rand ()/32767.0;

```

2. Position the cursor on the line following `amp = rand ()/32767.0`.
3. Expand the Library Tree to **Advanced Analysis Library** (or **Analysis Library**)»**Signal Generation**»**SinePattern**.
4. Enter 100 in the **Number of Elements** control.
5. Enter `amp` in the **Amplitude** control. Select **Code**»**Declare Variable** and make sure you enable the **Add declaration to current block in target file “sample4.c”** option. Click **OK**.
6. Enter 180.0 in the **Phase (Degrees)** control.
7. Enter 2.0 in the **Number of Cycles** control.
8. Enter `sine` in the **Sine Pattern** control. Select **Code**»**Declare Variable**.
9. In the **Declare Variable** dialog box, enter 100 as the **Number of Elements** and make sure you enable the **Add declaration to current block in target file “sample4.c”** option. Click **OK**.
10. Select **Code**»**Insert Function Call**. LabWindows/CVI inserts the `SinePattern` function after `amp = rand ()/32767.0;`.

Building the PlotY Function Call Syntax

Complete the following steps to generate a line of code that plots the random data array on the graph control on the user interface.

1. Position the cursor in the Source window on the line following the `SinePattern` function call within the `AcquireData` function.
2. Expand **User Interface Library**»**Controls/Graphs/Strip Charts**»**Graphs and Strip Charts**»**Graph Plotting and Deleting**»**PlotY**.

3. In the **Panel Handle** control, select **Code»Select Variable**. The dialog box contains a list of variable names used in your program. Choose **panelHandle** from the list and click **OK**. (If **panelHandle** is not listed, click **Build The Project**.)
4. For the **Control ID** control, you must specify the constant name assigned to the graph control. While the cursor is in **Control ID**, press <Enter> to open a dialog box with a complete list of all the constant names in the .uir files in the workspace. In the User Interface Resource Files section, select **\sample4.uir**. Select **PANEL_WAVEFORM** from the Select UIR Constant dialog box and click **OK**.
5. Type **sine** in the **Y Array** control. This name indicates which array in memory displays on the graph.
6. Type **100** in the **Number of Points** control. This number indicates the number of elements in the array to plot. When your Plot Y function panel matches the one in Figure 6-1, you are ready to go to the next step.

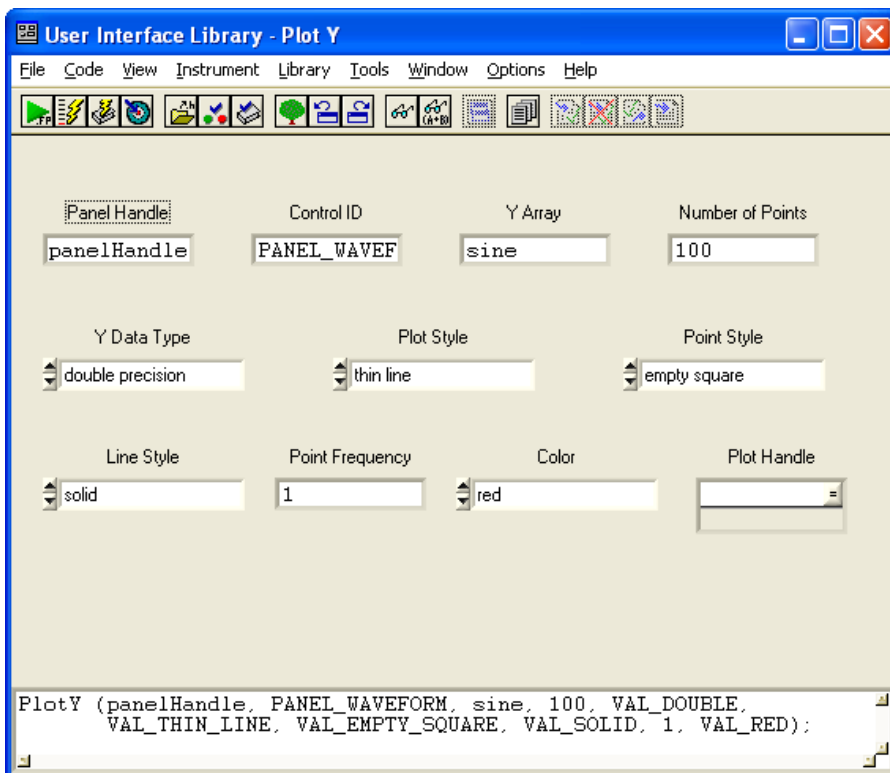


Figure 6-1. Completed Plot Y Function Panel

7. Select **Code»Insert Function Call** to paste the `PlotY` function call into the source code.

8. Confirm that the `AcquireData` function matches the following source code:

```
int CVICALLBACK AcquireData (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    double sine[100];
    double amp;
    switch (event) {
        case EVENT_COMMIT:
            srand (time(NULL));
            amp = rand ()/32767.0;
            SinePattern (100, amp, 180.0, 2.0, sine);
            PlotY (panelHandle, PANEL_WAVEFORM, sine, 100, VAL_DOUBLE,
                VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
            break;
        case EVENT_RIGHT_CLICK;

            break;
    }
    return 0;
}
```

9. Save the source file as `sample4.c`.

Running the Completed Project

You now have a completed project, saved as `sample4.prj`. Select **Run»Debug sample4_dbg.exe** to execute the code.

During the compile process, LabWindows/CVI recognizes that the program is missing the `ansi_c.h` and `analysis.h` include statements. Click **Yes** to add these include files in your program. If prompted, save the changes to the `sample4.c` file before running. When you run your program, the following steps take place:

1. LabWindows/CVI compiles the source code from `sample4.c` and links with the appropriate libraries in LabWindows/CVI.
2. When the program starts, LabWindows/CVI launches the user interface, ready for keyboard or mouse input.
3. When you click **Acquire**, LabWindows/CVI passes the event information generated by the mouse click directly to the `AcquireData` callback function.
4. The `AcquireData` function generates an array of random data and plots it on the graph control on the user interface.
5. When you click **QUIT**, the event information generated by the mouse passes directly to the `QuitCallback` function, which halts the program.

Adding Analysis to Your Program

In Chapter 6, *Using Function Panels and Libraries*, you generated code to plot the random array on the graph control. The plotting function that you generated was placed in a callback function triggered by clicking the **Acquire** button. In this chapter, you will add analysis code that computes the maximum and minimum values of the random array you generate. To do this, you will write your own callback function that finds the maximum and minimum values of the array and displays them in numeric readouts on the user interface.

This chapter builds on the concepts that you learned in the previous chapter. If you did not complete Chapter 6, *Using Function Panels and Libraries*, go back and do so now.

Setting Up

1. Close all windows except the Workspace window.
2. Make **Sample5** the active project.
3. Run `sample5.prj` to verify the operation of the program. Sample 5 matches the project you completed in Chapter 6, *Using Function Panels and Libraries*. Click **QUIT** to terminate the execution.

Modifying the User Interface

Complete the following steps to modify the existing user interface:

1. Open `sample5.c`. This code is similar to the resulting code from the previous example. Place the cursor at the end of the file. CodeBuilder uses that location for the new callback function that it generates later in this chapter.
2. Without closing the `sample5.c` source code, open `sample5.uir`. Your goal is to modify the `.uir` to match the user interface shown in Figure 7-1.

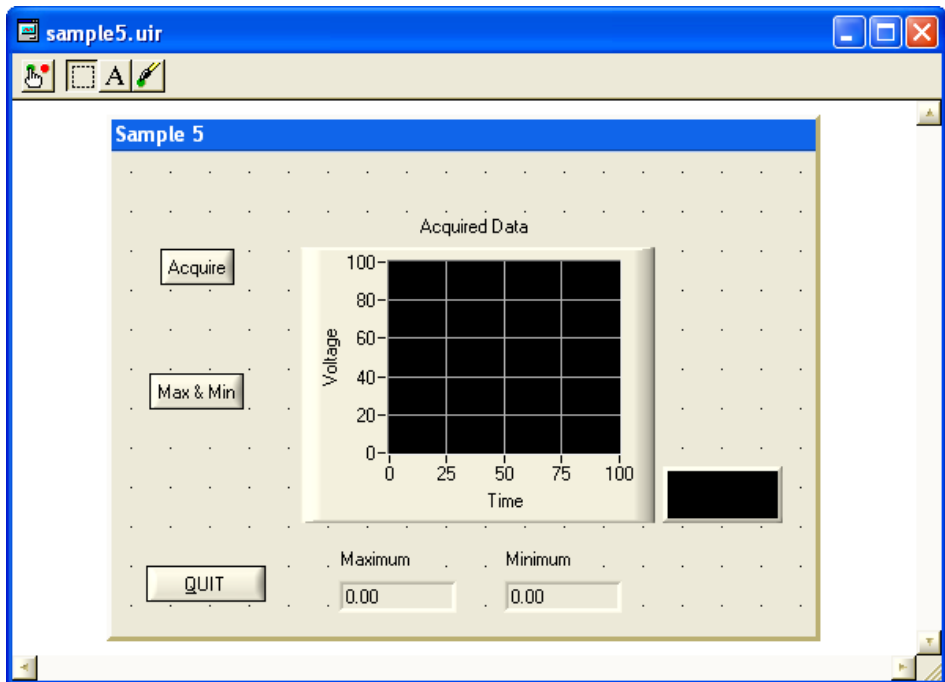


Figure 7-1. Sample User Interface

3. Add a command button to the panel.
4. Double-click the new command button to open the Edit Command Button dialog box. Enter the following information in the dialog box and click **OK**.

Constant Name: MAXMIN
Callback Function: FindMaxMin
Label: Max & Min

5. You can use CodeBuilder to add code to your program shell for an individual control callback function. Right-click the **Max & Min** command button and select **Generate Control Callback**.

The lightning bolt cursor appears while CodeBuilder generates code into the `sample5.c` source file. When you finish updating the user interface for Sample 5, you will add to the `FindMaxMin` callback function code to compute and display the maximum and minimum values of the array.

6. In the User Interface Editor, select **Create>Numeric>Numeric**.

7. Double-click the **Numeric** control to open the Edit Numeric dialog box. Enter the following information in the dialog box and click **OK**.

Constant Name: MAX
Control Mode: Indicator
Label: Maximum

8. Add a second numeric control to the panel.
9. Double-click the **Numeric** control to open the Edit Numeric dialog box. Enter the following information in the dialog box and click **OK**.

Constant Name: MIN
Control Mode: Indicator
Label: Minimum

10. Position the new controls on the user interface to match those shown in Figure 7-1.



Tip You can use the **Arrange»Alignment** command to position controls on the panel.

11. Save the modified `.uir` file.

Writing the Callback Function

Now that you have modified the `.uir` file and generated the shell for the callback function to the **Max & Min** command button, you need to complete the `FindMaxMin` function in the source file. Follow these steps.

1. To quickly locate the `FindMaxMin` callback function in your source file, right-click the **Max & Min** button in the User Interface Editor and select **View Control Callback**. LabWindows/CVI displays the `sample5.c` source file with the `FindMaxMin` callback function highlighted.
2. Position the cursor on the blank line just after the `case EVENT_COMMIT` statement. The code within the case statement executes when your program is running and you click the **Max & Min** button. You must enter function calls to find the maximum and minimum values of the `datapoints` array and display them on the user interface. Enter these function calls in the steps that follow.
3. Open the 1D Maximum & Minimum function panel by expanding the Library Tree to **Advanced Analysis Library»Array Operations»1D Operations»MaxMin1D**.



Note Remember, depending on which package you have, the Library Tree shows either the Analysis Library or the Advanced Analysis Library.

- The `MaxMin1D` function finds the maximum and minimum values of an array. Enter the following values into the controls on the function panel.

Input Array: `sine`
Number of Elements: `100`
Maximum Value: `max`
Maximum Index: `max_index`
Minimum Value: `min`
Minimum Index: `min_index`

- Before you insert the `MaxMin1D` function into the source code, you must declare the `max`, `max_index`, `min`, and `min_index` variables. Click the **Maximum Value** control and select **Code»Declare Variable**. Enable the **Add declaration to current block in target file “sample5.c”** option. Declaring the variable inserts a line of code to declare the `max` variable within the `FindMaxMin` callback function. Click **OK** to continue.



Note Notice that LabWindows/CVI automatically inserts an ampersand, `&`, before the `max` variable so that it is properly passed by reference to the function.

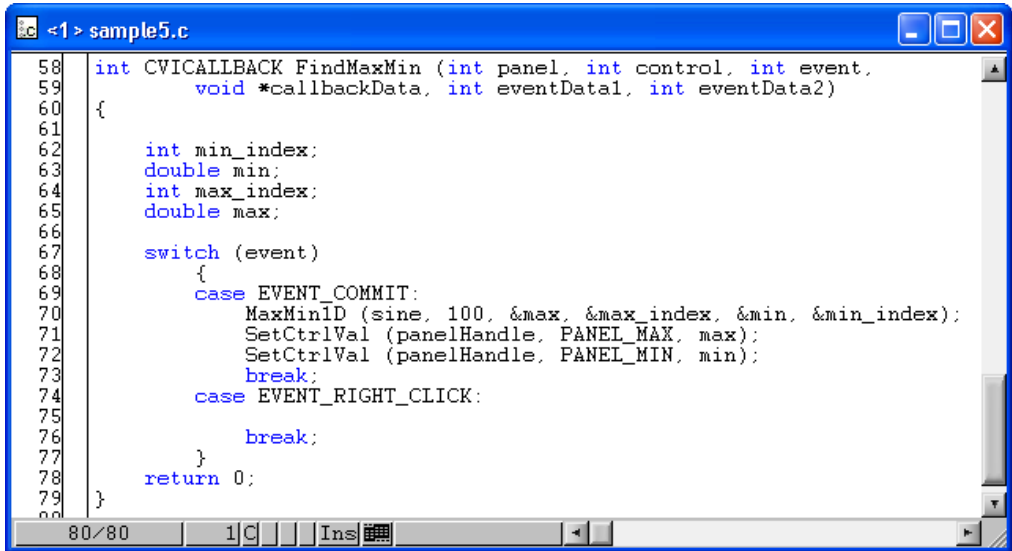
- Repeat step 5 for the **Maximum Index**, **Minimum Value**, and **Minimum Index** controls on the 1D Maximum & Minimum function panel.
- Insert the `MaxMin1D` function call into the source code. In the source code, LabWindows/CVI inserts the `MaxMin1D` function inside the `case EVENT_COMMIT` statement within the `FindMaxMin` callback function.
- Expand the Library Tree to **User Interface Library»Controls/Graphs/Strip Charts»General Functions»SetCtrlVal**.
- The `SetCtrlVal` function sets the value of a control on your user interface. Enter the following information into the function panel controls to display the maximum value of the array in the **Maximum** numeric control.

Panel Handle: `panelHandle`
Control ID: `PANEL_MAX`
Value: `max`

- Insert the `SetCtrlVal` function call into the source code. You then see the `SetCtrlVal` code entered on the line after the function call to `MaxMin1D` in the source code file.
- Open the Set Control Value function panel again by pressing `<Ctrl-P>` while the cursor is on the `SetCtrlVal` function.
- Enter the following information into the function panel controls to display the minimum value of the array in the **Minimum** numeric display.

Panel Handle: `panelHandle`
Control ID: `PANEL_MIN`
Value: `min`

13. Insert the `SetCtrlVal` function call into the source code. When LabWindows/CVI prompts you with a dialog box, select **Insert** to add the code.
14. Confirm that the source code matches the code shown in Figure 7-2.



```

58 int CVICALLBACK FindMaxMin (int panel, int control, int event,
59     void *callbackData, int eventData1, int eventData2)
60 {
61
62     int min_index;
63     double min;
64     int max_index;
65     double max;
66
67     switch (event)
68     {
69         case EVENT_COMMIT:
70             MaxMinID (sine, 100, &max, &max_index, &min, &min_index);
71             SetCtrlVal (panelHandle, PANEL_MAX, max);
72             SetCtrlVal (panelHandle, PANEL_MIN, min);
73             break;
74         case EVENT_RIGHT_CLICK:
75
76             break;
77     }
78     return 0;
79 }

```

Figure 7-2. Completed Source Code for sample5.c

Running the Program

You have now successfully written the callback function. During program execution, the `FindMaxMin` function is called when the program is running and you click **Max & Min**. When you click **Max & Min**, three separate events occur.

- First, clicking the left mouse button generates an `EVENT_LEFT_CLICK` event.
- Next, releasing the left mouse button generates an `EVENT_COMMIT` event. You wrote the function such that it finds the minimum and maximum values and displays them only when your program receives the `EVENT_COMMIT` event.
- Finally, the button gets the input focus, and an `EVENT_GOT_FOCUS` event is generated. For more practice with user interface events, complete [Exercise 5: Adding User Interface Events](#) of Chapter 9, *Additional Exercises*.

1. Save the file.
2. Run the project.
3. Close the file before moving on to Chapter 8, *Using an Instrument Driver*.

Using an Instrument Driver

This chapter describes how to use an instrument driver. An instrument driver is a set of functions used to program an instrument or a group of related instruments. The high-level functions in an instrument driver incorporate many low-level operations, including GPIB, VXI, or RS-232 read and write operations, data conversion, and scaling. The sample module in this chapter does not communicate with a real instrument but illustrates how you can use an instrument driver in your programs.

Setting Up

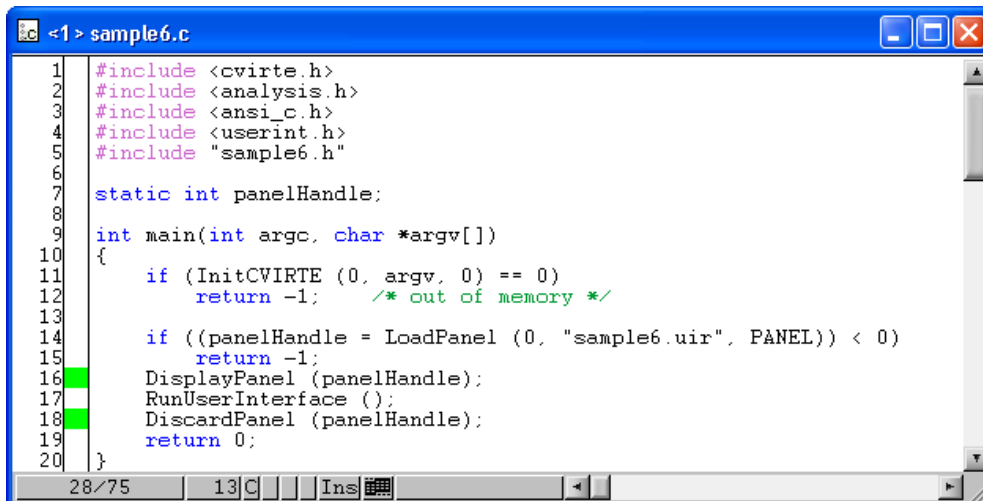
This section builds on the program that you created in Chapter 7, *Adding Analysis to Your Program*. If you have not completed the activities in that chapter, do so now.

Before beginning this example, close all windows except for the Workspace window and set `sample6.prj` as the active project.

Loading the Instrument Driver

An instrument driver consists of several files that reside on disk. Use the Instruments folder in the Library Tree to load these files for use in LabWindows/CVI. Complete the following steps to load the sample instrument driver.

1. Right-click the Instruments folder in the Library Tree and select **Load Instrument**.
2. In the Load Instrument dialog box, select the `scope.fp` file from the `tutorial` directory, and click **Load**.
3. Open `sample6.c`.
4. Position the cursor on the tagged line in the middle of the `main` function, on the `DisplayPanel` function, as shown in Figure 8-1. You can move to this line quickly in the source code by pressing <F2>.



```

1  #include <cvirte.h>
2  #include <analysis.h>
3  #include <ansi_c.h>
4  #include <userint.h>
5  #include "sample6.h"
6
7  static int panelHandle;
8
9  int main(int argc, char *argv[])
10 {
11     if (InitCVIRTE (0, argv, 0) == 0)
12         return -1; /* out of memory */
13
14     if ((panelHandle = LoadPanel (0, "sample6.uir", PANEL)) < 0)
15         return -1;
16     DisplayPanel (panelHandle);
17     RunUserInterface ();
18     DiscardPanel (panelHandle);
19     return 0;
20 }

```

Figure 8-1. sample6.c Source File with Tags

To verify that the Scope instrument driver was loaded, expand the Instruments folder. The **Sample Oscilloscope** item should appear in the tree, as shown in Figure 8-2.

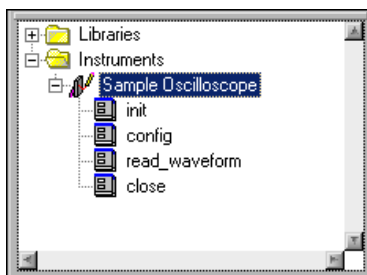


Figure 8-2. Sample Oscilloscope in Instruments Folder

Using the Instrument Driver

When you load the instrument driver, you can use it interactively through menus, dialog boxes, and function panels. Expand **Sample Oscilloscope** in the Instruments folder.

This module contains the `init`, `config`, `read_waveform`, and `close` functions. You will use these functions to acquire a waveform in the sample program you are developing.

Interactive Function Panel Execution

In Chapter 3, *Interactive Code Generation Tools*, you learned how to use function panels to generate code and insert that code into the programs you developed. You also can use function panels to execute the functions from the panel interactively, without writing a complete program. Therefore, you can experiment with functions by varying the parameter values in the panel and running the panels until you are satisfied with the result. Through trial and error, you can build your function calls in the function panel before inserting them into source code. In this section, you will learn how to execute function panels before inserting the code into the program.

Initializing the Instrument

Typically, instrument drivers use a function to initialize the software and the instrument. You must execute the initialize function before using any other function in the module. Select the Initialize function panel from the Instruments folder.

This function panel has an input control for specifying the GPIB address of the instrument. The **Error** control displays error codes related to the operation of this module.

You can access function help and control help in this instrument driver in the same way that you access help for LabWindows/CVI function panels.

Complete the following steps to initialize the instrument driver.

1. Enter 1 in the **Address** control.
2. Enter `err` in the **Error** control.
3. Declare the `err` variable by selecting **Code»Declare Variable**. Be sure to enable the **Execute declaration in Interactive Window** and **Add declaration to top of target file** options. Click **OK**.
4. Select **Code»Run Function Panel**. If LabWindows/CVI does not detect any errors during execution, the value in the **Error** control is 0.
5. Select **Code»Insert Function Call**, or press <Ctrl-I>, to copy the generated code to the Source window. If you are prompted for a target file, select `sample6.c`.

LabWindows/CVI inserts the function call to initialize the instrument driver above the `DisplayPanel` function call in the Source window as follows:

```
err = scope_init (1);
```

Configuring the Instrument

After you initialize the instrument, you can configure it to read a waveform and transfer the waveform to an array in your program. In the Sample Oscilloscope module, the vertical and

horizontal parameters of the oscilloscope are set up using the `config` function. Place the cursor on the `DisplayPanel` function. Select the Configure function panel.

The `config` function sets the volts per division and coupling of either Channel 1 or Channel 2 of the oscilloscope. This function also sets the horizontal time base of the instrument. The instrument driver is written to create a waveform based on the configuration settings. The help for each control explains the purpose of the control and the valid range of inputs.

Complete the following steps to execute the panel and save the code to your program.

1. Configure the panel the way you want it, keeping in mind that the way you configure the settings affects the waveform you read.
2. Enter `err` in the **Error** control.
3. Select **Code»Run Function Panel** to execute the function panel. If the **Error** control does not display a 0, an error has occurred.

If a real instrument were attached, you would be able to see the configuration of the instrument take place when you selected **Code»Run Function Panel**. Thus, you could interactively program the instrument and verify the operation of the instrument driver functions.

4. Insert the function into the code. LabWindows/CVI inserts the code after the call to `scope_init`.
5. Move the cursor to the blank line before the `PlotY` function in the `AcquireData` function. Leave the cursor in place as you go on to the next section.

Reading Data with an Instrument Driver

Perhaps the most important function of an instrument driver is to read data from an instrument and convert the raw data into a format your program can use directly. For example, a digital oscilloscope returns a waveform as a string of comma-separated ASCII numbers or as binary integers. In either case, the numbers are scaled using constants provided by the instrument to produce values that represent actual measurement units.

1. Select the Read Waveform function panel.
2. Set the **Channel** control to the channel you want to read. Channel 1 is a sinewave, and Channel 2 is random data.
3. Enter `datapoints` in the **Waveform** control.
4. Select **Code»Declare Variable**. In the Declare Variable dialog box, enter 100 as the **Number of Elements**. Enable the **Add declaration to top of target file “sample6.c”** option. Confirm that your Declare Variable dialog box matches the one shown in Figure 8-3.

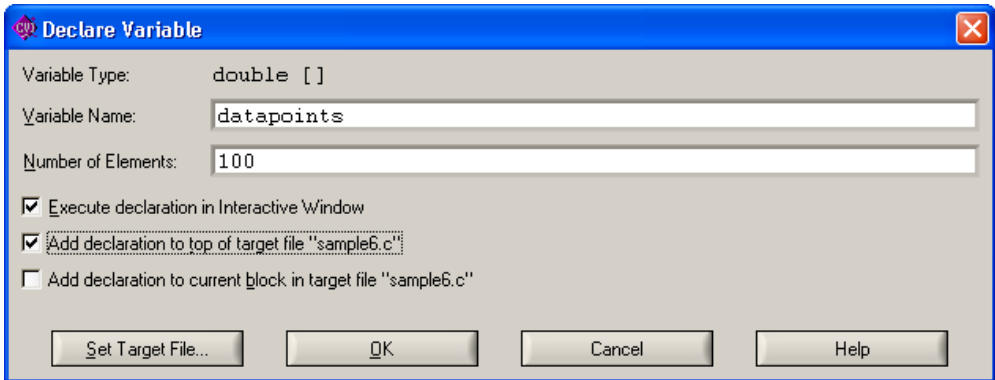


Figure 8-3. Declare Variable Dialog Box

5. Press <Enter> or click **OK** to declare the `datapoints` array.
6. Enter `delta_t` in the **Sample Period** control and select **Code»Declare Variable** to declare the `delta_t` variable.
7. Enter `x_zero` in the **X Zero** control and declare the `x_zero` variable.
8. Enter `err` in the **Error** control.
9. Select **Code»Run Function Panel** to execute the function panel. After the function has executed, a row of boxes in the **Waveform** control signifies that the data has been placed in the waveform array.
10. (Optional) To quickly view the data points acquired in the waveform array in the Array Display, double-click the row of boxes in the bottom half of the **Waveform** control on the function panel. Close the Array Display and the Variables windows.
11. Insert the function into the source code. LabWindows/CVI inserts the code before the `PlotY` function.

Closing the Instrument

The last instrument-related operation is closing the instrument driver. Complete the following steps to close the instrument driver.

1. In the Source window, position the cursor on the line following the `RunUserInterface` function. To do this quickly, press <F2>.
2. Select the Close function panel. There are no parameters for the `close` function. The `close` function removes the instrument from a software configuration table. The instrument must be reinitialized before using it again.
3. Enter `err` in the **Error** control.
4. Select **Code»Run Function Panel** to execute the function panel.
5. Select **Code»Insert Function Call** to copy the generated code to the Source window.

Running the Program

The last step required before running the program is to include the scope header file.

1. To call functions from the Scope instrument driver, you must add the following line at the top of the source file.

```
#include "scope.h"
```

2. Confirm that your program source code matches the following code:

```
#include <cvirte.h>
#include <analysis.h>
#include <ansi_c.h>
#include <userint.h>
#include "sample6.h"
#include "scope.h"

static double x_zero;
static double delta_t;
static double datapoints[100];
static int err;
static int panelHandle;

int main(int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return (-1);    /* out of memory */
    if ((panelHandle = LoadPanel (0, "sample6.uir", PANEL)) < 0)
        return -1;
    err = scope_init (1);
    err = scope_config (1, 1.0, 1, 1.0e-3);
    DisplayPanel (panelHandle);
    RunUserInterface ();
    err = scope_close ();
    DiscardPanel (panelHandle);
    return 0;
}

int CVICALLBACK AcquireData (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            err = scope_read_waveform (1, datapoints, &delta_t,
                &x_zero);
            PlotY (panelHandle, PANEL_WAVEFORM, datapoints, 100,
                VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
```



```

        VAL_SOLID, 1, VAL_RED);
        break;
    case EVENT_RIGHT_CLICK:

        break;
    }
    return 0;
}

int CVICALLBACK QuitCallback (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

int CVICALLBACK FindMaxMin (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    int min_index;
    double min;
    int max_index;
    double max;
    switch (event) {
        case EVENT_COMMIT:
            MaxMin1D (datapoints, 100, &max, &max_index, &min,
                &min_index);
            SetCtrlVal (panelHandle, PANEL_MAX, max);
            SetCtrlVal (panelHandle, PANEL_MIN, min);
            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

```



Note Your calls to `scope_config` and `scope_read_waveform` might differ from those shown in the preceding code.

3. Save `sample6.c`.
4. Run the program.

Adding the Instrument to Your Project

When you loaded the scope driver through the Instruments folder, you manually added the function panels of the instrument driver to LabWindows/CVI. If you add the Scope instrument driver to the project, LabWindows/CVI adds the instrument driver function panels to the Instruments folder automatically when you load the project in the future. Complete the following steps to add the driver to the project.

1. Close all windows except the Workspace window.
2. Select **Edit»Add Files to Project»Instrument (*.fp)**.
3. Add `scope.fp`.

Additional Exercises

This is the last chapter of the tutorial exercises. This chapter discusses more about the concepts you have used throughout this tutorial. Each exercise builds on the code that you develop in the preceding exercise.

You can access the solutions to all the exercises in this chapter in `\tutorial\solution`. If you have trouble completing one of the exercises but would like to continue to the next topic, use the solution from the previous exercise.

Base Project

All of the exercises in this chapter build on the Sample 6 project that you completed in Chapter 8, *Using an Instrument Driver*. If you did not complete the previous chapter, go back and do so now. If you have trouble successfully completing the Chapter 8 exercises, start with the Sample 6 project from the `solution` directory.

The Sample 6 project generates a waveform and displays it on a graph control when you click the **Acquire** button. After you display the data, you can find and display the maximum and minimum values of the data points by clicking the **Max & Min** button. The project uses the sample Scope instrument driver to generate the data. The user interface for the project is shown in Figure 9-1.

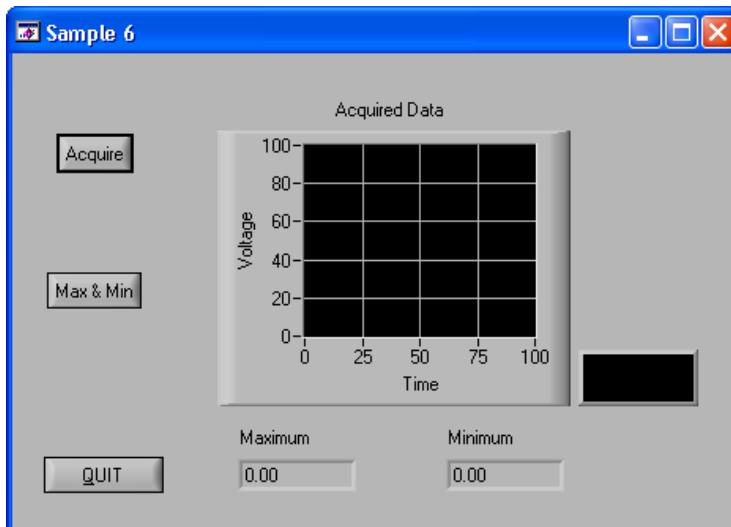


Figure 9-1. Sample User Interface

Exercise 1: Adding a Channel Control

Two of the most common functions you use in LabWindows/CVI are `SetCtrlVal` and `GetCtrlVal`. Use these functions to set and retrieve the current value of a control on a LabWindows/CVI `.uir` file. For example, use `GetCtrlVal` to retrieve the current value of a numeric slide control so that you can find out which selection the user has set the slide to. To set the slide control to a specific position or value, use `SetCtrlVal`. These functions take the following arguments:

- The panel handle for the panel where the control exists
- The control ID for the control to operate on
- A value or variable that the control is set to or in which the value of the control is placed

Assignment

Because you are using a simulated oscilloscope to acquire your data, you might want to give the end user of your program the ability to select the channel from which to acquire the data. The sample oscilloscope driver can read from two channels. To successfully complete this exercise, you must modify the `.uir` file of the base project, Project 6, to include a channel selection control, as shown in Figure 9-2, and modify the source code to properly acquire the correct channel.

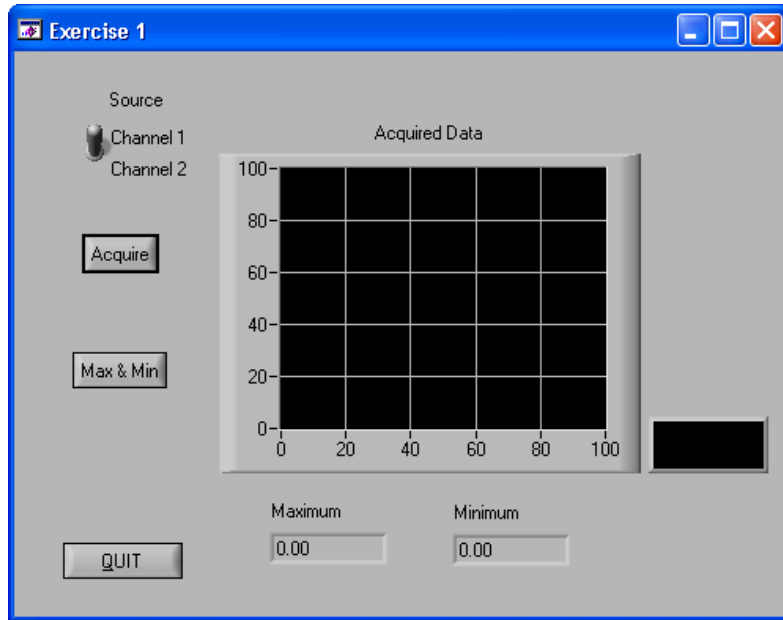


Figure 9-2. User Interface with Channel Selection Control

Hints

- Use a binary switch for the channel selection control.
- Use the `GetCtrlVal` function in the `AcquireData` callback function to find out which channel the user selects.
- Use the value from the channel selection control in the `read_waveform` function call from the Scope instrument driver.

Solution: `exer1.prj`

Exercise 2: Setting User Interface Attributes Programmatically

Each control on the `.uir` files that you create has a number of control attributes that you can set to customize the look and feel of the control. When you build a user interface, you set the control attributes in the dialog boxes for editing the controls. For example, you can set the font, size, and color of the text for a label in the User Interface Editor. Text font, size, and color are user interface control attributes.

Use `GetCtrlAttribute` and `SetCtrlAttribute` to get and set attributes of a control during program execution in a method similar to the one you used to get and set the value of a control. Therefore, you can build a customized GUI in the User Interface Editor and dynamically change the look and feel of the controls at run time.

Hundreds of attributes are pre-defined in the User Interface Library as constants, such as `ATTR_LABEL_BGCOLOR` for setting the background color of the label on a control. You can use these constants in the `GetCtrlAttribute` and `SetCtrlAttribute` functions.

Assignment

In this exercise, use the `SetCtrlAttribute` function to change the operation of a command button on the user interface. Because the **Max & Min** command button does not operate correctly until you acquire the data, you can disable the **Max & Min** button until a user clicks the **Acquire** button. Use the `SetCtrlAttribute` function to enable the **Max & Min** button when a user clicks the **Acquire** button.

Hints

- Start by disabling (dimming) the **Max & Min** command button in the User Interface Editor.
- Use the `SetCtrlAttribute` function from the User Interface Library to enable the **Max & Min** button.
- The attribute that you need to set is the *dimmed* attribute.

Solution: `exer2.prj`

Exercise 3: Storing the Waveform on Disk

Many times, users acquire large amounts of data and want to save it on disk for future analysis or comparison. LabWindows/CVI provides a selection of functions from the ANSI C Library for reading from and writing to data files. If you are already familiar with ANSI C, you know these functions as the `stdio` library. In addition to the `stdio` library, LabWindows/CVI has its own set of file I/O functions in the Formatting and I/O Library.

Assignment

Use the file I/O functions in the ANSI C Library to save the `datapoints` array to a text file in the `\tutorial` directory. Write the program so that the file is overwritten each time you acquire the data. Do not append data to the file as you acquire it.

Hints

- Remember that you must first open a file before you can write to it.
- Open the file as a text file so you can view the contents in any text editor later.
- Open the file with the Create/Open flag and not the Append flag so that the file is overwritten each time.
- Use the `fprintf` function in a loop to write the data to disk.

Solution: `exer3.prj`

Exercise 4: Using Pop-up Panels

The User Interface Library has a set of predefined panels called pop-up panels. Pop-up panels provide a quick and easy way to display information on the screen without developing a complete `.uir` file. In Chapter 5, *Building a Graphical User Interface*, you used a pop-up panel to display the random number array on a graph (`YGraphPopup`). You also can use pop-up panels to prompt the user for input, confirm a selection, or display a message.

One of the most useful pop-up panels is the File Select Popup. With the File Select Popup, you can use a File Save or File Load dialog box within the programs you develop in LabWindows/CVI. Therefore, whenever your program must write to a file or read from a file, you can use the File Select Popup, shown in Figure 9-3, to prompt the user to select or input a filename.

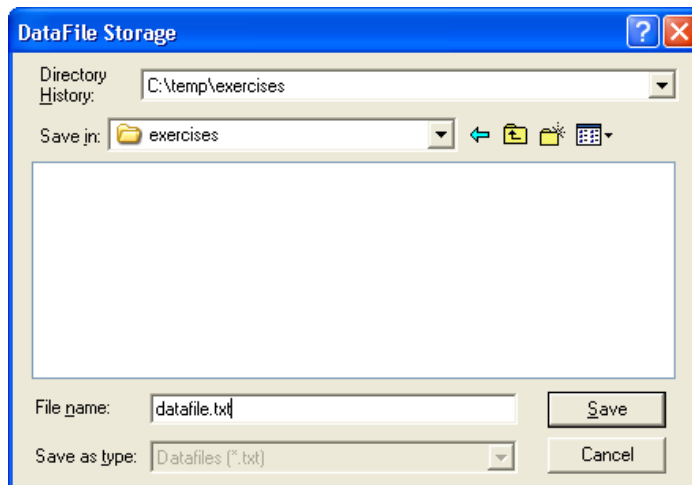


Figure 9-3. File Select Popup

Assignment

Add a **Save** button to the `.uir` file so that the data in the array is saved only after the user clicks the **Save** button. When the user clicks the **Save** button, your program should launch a dialog box in which the user can define the drive, directory, and filename of the data file. When finished, the `.uir` file should look similar to the one shown in Figure 9-4.

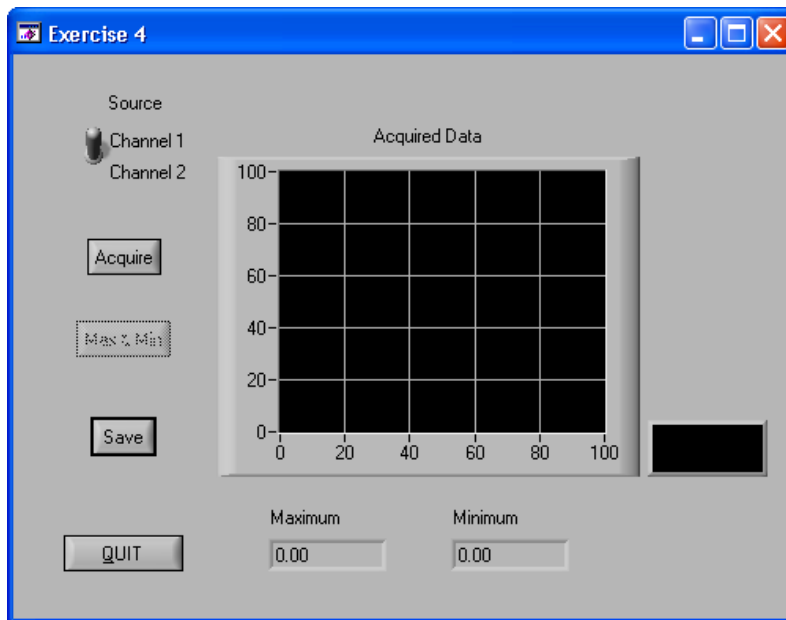


Figure 9-4. Completed User Interface

Hints

- When you create the **Save** button, assign a callback function to it.
- You must move the source code that you developed in Exercise 3 for writing the array to disk into the callback function.
- Before you write the data to disk, prompt the user for a filename with the `FileSelectPopUp` function from the User Interface Library.

Solution: `exer4.prj`

Exercise 5: Adding User Interface Events

Throughout this tutorial, you have been developing an event-driven program. When you place a control on a `.uir` file, you are defining a region of the screen that can generate events during program execution. Your C source files are written to respond to these events in callback functions.

So far, you have written functions that respond only to the `COMMIT` event from the user interface. A `COMMIT` event occurs whenever the end user commits on a control, which usually happens when that user releases the left mouse button after clicking a control.

User interface controls can generate many different types of events. For example, an event can be a left-click or a right-click. Or, an event can be a left double-click. In fact, events in LabWindows/CVI can be more than just mouse clicks. An event can be the press of a key or a move or size operation performed on a panel. Each time one of these events occurs, the callback function associated with the user interface called executes. LabWindows/CVI can generate the following events:

<code>EVENT_CLOSE</code>	<code>EVENT_LEFT_CLICK</code>
<code>EVENT_COLLAPSE</code>	<code>EVENT_LEFT_DOUBLE_CLICK</code>
<code>EVENT_COMMIT</code>	<code>EVENT_LOST_FOCUS</code>
<code>EVENT_DISCARD</code>	<code>EVENT_MARK_STATE_CHANGE</code>
<code>EVENT_DRAG</code>	<code>EVENT_PANEL_MOVE</code>
<code>EVENT_DROP</code>	<code>EVENT_PANEL_SIZE</code>
<code>EVENT_DROPPED</code>	<code>EVENT_RIGHT_CLICK</code>
<code>EVENT_END_TASK</code>	<code>EVENT_RIGHT_DOUBLE_CLICK</code>
<code>EVENT_EXPAND</code>	<code>EVENT_SELECTION_CHANGE</code>
<code>EVENT_GOT_FOCUS</code>	<code>EVENT_SORT</code>
<code>EVENT_HSCROLL</code>	<code>EVENT_TIMER_TICK</code>
<code>EVENT_IDLE</code>	<code>EVENT_VAL_CHANGED</code>
<code>EVENT_KEYPRESS</code>	<code>EVENT_VSCROLL</code>

When the callback function is called, the event type is passed through the event parameter to the callback function. Performing one simple operation on the user interface, such as clicking a command button, actually calls the callback function for that button three times.

The first time, the callback function is called to process the `EVENT_LEFT_CLICK` event. The second time, it is called to process the `EVENT_COMMIT` event. The third time, the callback function is called to process the `EVENT_GOT_FOCUS` event if the button did not have the input focus before you clicked it. For this reason, all of the callback functions you have worked on check the event type first and execute only when the event is a `COMMIT`. Therefore, the operations in the callback functions happen only once with each event click, rather than three times.

Assignment

Many times, the person operating a LabWindows/CVI program is not the person who developed the program. The GUI might be very easy to use, but usually it is preferable to add help for the controls on `.uir` panels to assist the operator. Modify `exer4.prj` to display a short description for each command button when the user right-clicks the button.

Hints

- Use the `MessagePopup` function to display the help.
- Remember that the event type is passed to each callback function in the event parameter.
- The event that you must respond to is `EVENT_RIGHT_CLICK`.



Tip If you want to add pop-up documentation to controls, use `SetCtrlToolTipAttribute`. You can find this function by expanding the Library Tree to **Programmer's Toolbox»User Interface Utilities»SetCtrlToolTipAttribute**.

Solution: `exer5.prj`

Exercise 6: Timed Events

You have developed an event-driven program that responds to events generated by mouse clicks or keypresses from the user. With the LabWindows/CVI timer control, you can generate events at specified time intervals to trigger program actions without requiring an action from the user.

You can include timer controls in your program by creating them in the User Interface Editor. The timer control is visible only at design time in the User Interface Editor. At run time, the timer control does not appear. You can specify a constant name, callback function, and timer event interval in the Edit Timer dialog box. LabWindows/CVI automatically calls the specified timer callback function with an event of type `EVENT_TIMER` each time the specified time interval elapses. The interval value is specified in seconds with a resolution of 1 millisecond between timer events.

Assignment

Add a thermometer control to the user interface and use a timer control to generate a random number and display it on the thermometer once each second.

Hints

- Set the timer interval to 1.
- Use CodeBuilder to generate the shell for the timer control callback function.
- Use `SetCtrlVal` to display the random number on the thermometer.

Solution: `exer6.prj`

Instrument Control and Data Acquisition

- Chapter 10, *Getting Started with GPIB and VXI Instrument Control*, is a quick reference to help you install and configure your IEEE 488.2 Interface board or VXI controller for use with LabWindows/CVI. The information included in this chapter is presented in more detail in the documentation that you receive with your hardware.
- Chapter 11, *Getting Started with Data Acquisition*, is a quick reference for installing and configuring National Instruments plug-in data acquisition (DAQ) devices for use with LabWindows/CVI. This chapter discusses how to install and configure both hardware and software and how to test the board operation. The information included in this chapter is presented in more detail in the documentation that you receive with your DAQ hardware and NI-DAQmx and Traditional NI-DAQ software.

Getting Started with GPIB and VXI Instrument Control

This chapter is a quick reference to help you install and configure your IEEE 488.2 Interface board or VXI controller for use with LabWindows/CVI. For more information about the contents included in this chapter, refer to the documentation that you receive with your hardware.

Getting Started with the GPIB Controller

The following sections include an introduction to GPIB and instructions for installing a GPIB interface board, configuring the software, and developing your applications.

Introduction to GPIB

The General Purpose Interface Bus (GPIB) is a bus protocol for controlling stand-alone, rack-and-stack instruments from external computers. Also known as the IEEE 488 standard, GPIB simplifies the interconnection of programmable instruments by defining the electrical, mechanical, and functional specifications for instrument controllers and talker/listener devices. IEEE 488 is now referred to as IEEE 488.1-1987.

In 1992, the IEEE 488.2 specification was created to further standardize the way instruments and controllers operate. IEEE 488.2 defines control sequences, common data formats, status reporting, and common commands for GPIB instrument control.

National Instruments GPIB controller hardware and software follow the IEEE 488.1, IEEE 488.2, and HS488 specifications for controllers. The National Instruments TNT4882C and NAT4882 GPIB controller ASICs continue to improve and advance GPIB communication. Both the TNT4882C and NAT4882 GPIB controllers are IEEE 488.2 compatible.

National Instruments has designed a high-speed data transfer protocol for IEEE 488 called *HS488*. This protocol increases performance for GPIB reads and writes up to 8 Mbytes/s, depending on your system.

Installing the GPIB Interface Board

LabWindows/CVI works with various National Instruments GPIB interfaces. First, verify the GPIB driver version you have. Open NI Measurement & Automation Explorer (MAX) and expand **My System»Software»NI-488.2**. After you verify your driver version, refer to ni.com/support/gpib/versions.htm to view a list of GPIB interfaces.

Each hardware kit comes with detailed information about configuring and installing GPIB hardware.

Configuring the GPIB Driver Software

NI-488.2 is more than just a library of routines for controlling GPIB instruments. NI-488.2 includes a number of software utilities for testing and configuring the operation of the controller, including the following utilities:

- A configuration utility for setting the interrupts, DMA channels, and general configuration information for your GPIB interface
- An interactive control program for executing functions over GPIB that you enter from the keyboard
- A bus monitoring utility that displays the bus activity during GPIB communication

The documentation that you receive with your GPIB software describes these and other utilities.

Configuring LabWindows/CVI for GPIB

LabWindows/CVI uses the NI-488.2 DLL for Windows that is included with your National Instruments GPIB interface hardware. You must configure LabWindows/CVI to load the GPIB libraries and associated function panels into the LabWindows/CVI programming environment. To load the GPIB libraries, select **Library»Customize** and select the **GPIB/GPIB 488.2** option.

Developing Applications

LabWindows/CVI contains function panels for generating code and executing function calls from the IEEE 488/488.2 Library. These function panels access the library functions from the NI-488.2 Library that come with your GPIB controller. This library is a DLL. While the function panels in LabWindows/CVI provide function panel help for using these functions, you can find detailed function descriptions for the GPIB/GPIB 488.2 Library functions in the online documentation that you receive with your GPIB software.

Getting Started with the VXI Controller

The following sections include an introduction to VXI; information about the VXI development system; and instructions for installing and configuring VXI hardware, configuring VXI software, developing applications, and using instrument drivers.

Introduction to VXI

VME eXtensions for Instrumentation (VXI) is a platform for instrumentation systems. VXI is used in a wide variety of test, measurement, instrument control, and automated test equipment (ATE) applications. VXI also is experiencing growth as a platform for data acquisition and analysis in research and industrial control applications.

VXI uses a mainframe chassis with a maximum of 13 slots to hold modular instruments or plug-in devices. Because VXI is based on the VMEbus standard, you also can use VME modules in VXI systems. The VXI backplane combines the 32-bit VME computer bus and high-performance instrumentation buses for precision timing and synchronization between instrument components.

You can control VXIbus instruments through three different types of controllers: embedded VXI computers, external MXI controllers installed in a standard PC or workstation, or IEEE 488.2 controllers from a PC or workstation.

VXI Development System

The LabWindows/CVI development system contains software for controlling VXI instruments for any of the methods described in the section *Introduction to VXI*.

The VXI controller contains low-level driver software called NI-VXI. NI-VXI includes a standard library of functions and utility programs for controlling and configuring the VXI bus. You must install the NI-VXI driver software in addition to the LabWindows/CVI VXI Library to control VXI instruments.

Installing and Configuring VXI Hardware

LabWindows/CVI works with the following VXI controllers:

- VXIpc Model Series
- VXI-PCI MXIbus Series
- MXI-2 Based VXI/VME Controllers for PXI/PCI
- MXI-3 Based VXI Controllers for PXI/PCI
- FireWire-based VXI Controllers
- GPIB-VXI/C

Each one of these controllers has documentation for installing and configuring the appropriate VXI hardware and software. For more information about installing and configuring VXI hardware, refer to the getting started manuals for the controller.

Configuring VXI Driver Software

NI-VXI is more than just a library of routines for controlling VXIbus instruments. NI-VXI, like NI-488.2, contains configuration and troubleshooting utility software for your VXIbus system, including the following utilities:

- **Measurement & Automation Explorer (MAX)**—A program you can use to configure hardware; add new channels, interfaces, and virtual instruments; execute system diagnostics; and view the devices and instruments connected to your system.
- **NI-Spy**—A utility you can use to track the calls your application makes to NI test and measurement drivers.
- The National Instruments multimainframe Resource Manager.
- **VISAIC**—An interactive control program that executes NI-VISA functions that you enter with a convenient graphical interface. National Instruments recommends that you use NI-VISA as the programming interface to control VXI systems.

Configuring LabWindows/CVI for VXI

LabWindows/CVI uses the NI-VXI DLL for Windows that is included with your National Instruments VXI controller hardware. You must configure LabWindows/CVI to load the VXI libraries and associated function panels into the LabWindows/CVI programming environment. To include the VXI libraries, select **Library»Customize** and select the **VXI** option.

Developing Applications

LabWindows/CVI contains function panels for generating code and executing function calls from the VXI Library. These function panels access the library functions from the NI-VXI Library that come with your VXI controller. This library is a DLL. The function panels in LabWindows/CVI provide help about using these functions, but you also can refer to the *NI-VXI API Reference Help* for more information about VXI Library functions.

Using Instrument Drivers

Instrument control with LabWindows/CVI is simplified tremendously with LabWindows/CVI. You can use instrument drivers to control GPIB, serial, Ethernet, and VXIbus instruments. Instrument drivers are custom libraries written to control specific instruments at a high level. Instead of learning all the low-level command sequences and syntax for your instruments, you can use an instrument driver that builds these command sequences based on inputs from the driver function panels. Therefore, you can communicate

with your instrument using intuitive, high-level steps, such as Initialize, Configure, and Measure.

LabWindows/CVI instrument drivers are available to you in source code so you can optimize the driver to work best for your application.

If you plan to use instrument drivers in your application, refer to Chapter 8, *Using an Instrument Driver*. If you plan to develop an instrument driver yourself, refer to the *LabWindows/CVI Instrument Driver Developers Guide* to learn how to use the development tools such as the IVI wizard for creating function trees, function panels, and instrument control source code for a driver.

Getting Started with Data Acquisition

This chapter is a quick reference for installing and configuring National Instruments plug-in data acquisition (DAQ) devices for use with LabWindows/CVI. This chapter discusses how to install and configure both hardware and software and how to test the operation of a device. For more information about the contents included in this chapter, refer to the documentation that you receive with your DAQ hardware and NI-DAQmx software.

Introduction to Data Acquisition

By using a plug-in DAQ device with LabWindows/CVI, you can acquire analog and digital signals directly into computer memory. National Instruments DAQ devices are available in many configurations and options. The most common type of DAQ system is a multifunction device, which has analog I/O, digital I/O, and counter/timer capabilities. For more specialized applications, DAQ devices are available with high-precision analog inputs, high-speed analog inputs, more digital I/O lines, or multiple counter/timers.

Applications for plug-in DAQ devices range from simple temperature measurement to complex process control systems. You can use a DAQ device to take single-point voltage readings or high-speed waveform acquisitions. You can configure your device to multiplex through many input channels at high speed or trigger complex acquisition algorithms with the onboard counter/timers. With LabWindows/CVI and a DAQ device, you can configure your system easily to match the specific needs of your application.

The driver software for controlling DAQ devices, NI-DAQmx and Traditional NI-DAQ, is included in the National Instruments Driver CD that comes with the LabWindows/CVI installation. LabWindows/CVI automatically loads the library of functions for controlling a National Instruments DAQ device if you install NI LabWindows/CVI Support for either NI-DAQ API on your PC.

Installing Software

The NI-DAQmx driver software that controls a National Instruments DAQ device contains functions for performing I/O with the device and utilities for resource management and for data and buffer management.

When you install LabWindows/CVI, select the Device Drivers feature on the LabWindows/CVI installation disk. To install the NI-DAQ software, select the API you want to use under **Data Acquisition** on the Driver CD. If you select **NI LabWindows/CVI Support** from the NI-DAQ setup program, all the files that are required for DAQ operations are installed.

Configuring Your Device for Data Acquisition

Before you can launch LabWindows/CVI and begin programming, you must first configure the DAQ device using MAX.

Testing the Operation of the Device and Configuration

At this point, you have configured and installed the DAQ device, installed LabWindows/CVI and the DAQ software, and configured the software with MAX. Before you start writing programs in LabWindows/CVI, try some simple testing to make sure that the device is installed and functioning correctly. You can perform two levels of simple testing for the DAQ device.

- **MAX**—When you configure your system, you perform the first level of testing on the device. From MAX, expand **Devices and Interfaces** in the Configuration window. Find your device under **NI-DAQmx Devices**. Right-click the device and select **Test Panels**.
- **DAQ Assistant**—In LabWindows/CVI, select **Tools»Create/Edit DAQmx Tasks** to create a new task using the DAQ Assistant. The DAQ Assistant walks you through the process of creating a task. Once you have created a task, click **Test**.

Developing Applications

When you know that the DAQ device is properly communicating with the software, you can begin developing applications. You can find a set of example programs for performing common DAQ tasks when you install the NI-DAQmx software. Refer to the `samples` folder and the NI Example Finder, which is accessible through **Help»Find Examples**. These tools and examples are a good starting point for applications.

DAQ Assistant

The DAQ Assistant provides a graphical interface to help you configure tasks and channels. You can use the DAQ Assistant to generate NI-DAQmx code to run tasks. The DAQ Assistant contains help for each step as you create or edit a DAQmx task.

Data Acquisition Library Sample Programs

The NI-DAQ for Windows software installs example programs into the `\samples\DAQmx` directory.

Related Documentation

The following help or manuals contain additional information about data acquisition.

- *NI-DAQmx Help*
- *DAQ Quick Start Guide*
- *DAQ Assistant Help*
- *Traditional NI-DAQ Function Reference Help*



Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
 - **Self-Help Resources**—For immediate answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
 - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

A

active window	The window affected by keyboard input at a given moment. The title of an active window appears highlighted.
API	Application programming interface.
Array Display	A window for viewing and editing numeric arrays.
ATE	Automated test equipment.

B

binary control	A function panel control that resembles a physical on/off switch and can produce one of two values depending on the position of the switch.
breakpoint	An interruption in the execution of a program. Also, a function in code that causes an interruption in the execution of a program.

C

checkbox	A dialog box item that allows you to toggle between two possible execution options.
click	A mouse-specific term; to quickly press and release the mouse button.
CodeBuilder	The LabWindows/CVI feature that creates code based on a <code>.uir</code> file to connect a GUI to the rest of a program. You can compile and run this code as soon as it is generated.
command button	A dialog box item that, when selected, executes a command associated with the dialog box.
control	Function panel: An input or output device that appears on a function panel for specifying function parameters and displaying function results. User interface: An object, on a panel, that displays information or accepts input from a user.

cursor The flashing rectangle that shows where you can enter text on the screen. There is also a rectangular mouse cursor, or pointer, that shows the position of the mouse.

D

Debugging Region An area of the Workspace window that contains the Variables, Watch, and Memory windows.

default command The action that takes place when <Enter> is pressed and no command is specifically selected. Default command buttons in dialog boxes have an outline around them.

dialog box A prompt mechanism in which you specify additional information needed to complete a command.

DLL Dynamic link library.

double-click A mouse-specific term; to click the mouse button twice in rapid succession.

drag A mouse-specific term; to hold down the mouse button while moving the mouse across a flat surface, such as a mouse pad.

F

.fcp file A file that contains information about the function tree and function panels of an instrument driver.

function panel A screen-oriented user interface to the LabWindows/CVI libraries that allows interactive execution of library functions and is capable of generating code for inclusion in a program.

function tree The hierarchical structure in which the functions in instrument drivers and LabWindows/CVI libraries are grouped.

G

Generated Code pane A small window located at the bottom of the screen that displays the code produced when you add values to function panel controls.

global control A function panel control that displays the value of a global variable within a function.

GPIB	General Purpose Interface Bus.
Graphical Array Display	A window in which you can view the values of arrays in a graph.
GUI	Graphical user interface.

H

highlight	To make a LabWindows/CVI screen item ready for input.
HS488	A high-speed version of the IEEE 488 bus that provides speeds of up to 8 Mbytes/s. HS488 uses a noninterlocked handshake protocol to transfer data among two or more devices. By using the HS488 protocol, devices can transfer data at rates that are higher than the rates that are possible by using the IEEE 488.2 protocol.

I

input control	A function panel control in which a value or variable name is entered from the keyboard.
instrument driver	A group of several subprograms related to a specific instrument that reside on disk in a special language-independent format. An instrument driver is used to generate and execute code interactively through menus, dialog boxes, and function panels.
Interactive Execution window	A LabWindows/CVI work area in which sections of code can be executed without creating an entire program.

L

Library Tree	An area in the Workspace window that contains a tree view of the LabWindows/CVI libraries and instruments.
list box	A dialog box item that displays a list of possible choices for completing a command.

M

menu	An area accessible from the menu bar that displays a subset of the possible menu items.
mouse cursor	A mouse-specific term; the rectangular block on the screen that shows the current mouse position.

O

output control	A function panel control that displays the results of a function.
Output Region	An area of the Workspace window in which errors, output, and search match windows appear.

P

point	A mouse-specific term; to move the mouse until the pointer rests on the item you want to click on.
pointer	A mouse-specific term; the rectangular block on the screen that shows the current mouse position.
press	A mouse-specific term; to hold down the mouse button.
project	A list of files, usually including a source file, user interface resource file, and header file, that your application uses.
Project Tree	An area of the Workspace window that contains the lists of projects and files in the current workspace.

R

return value control	A function panel control that displays a function result returned as a return value rather than as a formal parameter.
ring control	A control that displays a list of options one option at a time. Ring controls appear on function panels and in dialog boxes.

S

scroll bars	Areas along the bottom and right sides of a window that show your relative position in the file. You can use the scroll bars to move about in the window if you have a mouse installed.
scrollable text box	A dialog box item that displays text in a scrollable display.
select	To choose the item that the next executed action will affect by moving the input focus (highlight) to a particular item or area.
shortcut key commands	A combination of keystrokes that automatically executes a command.
slide control	A function panel control that resembles a physical slide switch and inserts a value in a function call that depends on the position of the cross-bar on the switch.
Source window	A LabWindows/CVI work area in which you edit and execute complete programs. The file extension <code>.c</code> designates a file that appears in this window.
Standard Input/Output window	A LabWindows/CVI work area in which output to and input from the screen take place.
standard libraries	The LabWindows/CVI ActiveX, Advanced Analysis (or Analysis), DDE Support, DIAdem Connectivity, Formatting and I/O, GPIB/GPIB-488.2, RS-232, TCP Support, User Interface, and Utility libraries and the ANSI C Library.
step mode	A program execution mode in which a program is manually executed one instruction at a time. Each instruction in the program is highlighted as it is executed.
String Display	A mechanism for viewing and editing string variables and arrays.
subwindow	A Source window, split into two scrollable editing areas for the same file.

T

text box	A dialog box item in which the user enters text from the keyboard to complete a command.
----------	------------------------------------------------------------------------------------------

timer control A user interface control that schedules the periodic execution of a callback function. A typical use of this control might be to update a graph every second.

tooltip A small, yellow box that displays the value of variables and expressions in a Source window.

U

User Interface Editor A graphical drag-and-drop editor for designing user interfaces for programs.

User Interface Editor window A window that displays the graphical representation you have designed for your project.

User Interface Library Includes a set of functions for controlling the interface programmatically and a resource editor for defining the user interface components.

V

Variables window A display that shows the values of variables currently defined in LabWindows/CVI.

VXI VME eXtensions for Instrumentation (bus)

W

window A working area that supports operations related to a specific task in the development and execution processes.

Window Confinement Region An area of the Workspace window that contains open Source, User Interface Editor, and Function Tree Editor windows, and function panels.

workspace A file that contains settings that do not affect the way a project builds, such as breakpoints, window position, tag information, and debugging levels. A workspace can contain more than one project.

Workspace window The main work area in LabWindows/CVI; contains the Project Tree, Library Tree, Window Confinement Region, Debugging Region, and Output Region.

Index

A

- accessing function panels, 2-2, 3-2
- Add Files to Project command, Edit Menu
 - adding instruments, 8-8
 - constructing projects, 6-5
- Add Watch Expression command, Options menu, 4-9
- ANSI C
 - ANSI C Library, 1-3
 - ANSI C specifications, 2-4
- Array Display window
 - displaying arrays, 4-7
 - editing arrays, 4-8
- arrays
 - declaring from function panels, 6-3
 - displaying, 4-7
 - displaying in graph, 4-10
 - editing, 4-8
 - generating random array of data, 6-3
- attributes, setting programmatically, 9-3

B

- binary switch controls,
 - adding to user interface, 9-3
- Break on First Statement command,
 - Run menu, 4-1, 4-5, 4-8
- breakpoints
 - breakpoint on error, 4-3
 - conditional, 4-3
 - definition, 4-3
 - instant breakpoints, 4-3
 - manual breakpoints, 4-4
 - predetermined breakpoints, 4-3
 - programmatic breakpoints, 4-3
 - watch expression breakpoints, 4-3

- Build Errors command, Window menu, 2-4
- Build Errors window, 2-4

C

- C language, using in LabWindows/CVI, 1-4
- callback functions
 - adding with CodeBuilder, 7-2
 - locating with CodeBuilder, 7-3
 - processing events (example), 9-7
 - writing, 7-3
- code generation
 - automatic. *See* CodeBuilder
 - interactive. *See* interactive code generation tools
- Code menu
 - Function Panel windows
 - Declare Variable command, 7-4, 8-4
 - Insert Function Call command, 3-6, 6-4, 8-3, 8-4, 8-5
 - Run Function Panel command, 8-3, 8-5
 - Set Target File command, 3-6
- code. *See* source files
- CodeBuilder
 - adding control callback function, 7-2
 - generating program shell, 1-5, 5-6
- commit events, 7-5, 9-7
- compile error messages, 2-4
- conditional breakpoints, 4-3
- configuration
 - data acquisition boards
 - LabWindows/CVI, 11-2
 - testing, 11-2
 - GPIB driver software, 10-2
 - instrument drivers, 8-3
 - VXI driver software, 10-4
 - VXI hardware, 10-3

- constants, displaying in .uir file, 6-4
- Continue command, Run menu (table), 4-2
- Control Callback Events dialog box, 5-6
- controls
 - binary switch controls, 9-3
 - command button control, 5-3, 7-2
 - graph control, 5-4
 - numeric control, 7-3
 - numeric slide control, 9-2
 - ring control, 3-5
 - timer control, 9-8
- conventions used in manual, *ix*
- Customize Library command, Library menu
 - loading GPIB libraries, 10-2
 - loading the VXI libraries, 10-4
- customizing toolbars, 2-3

D

- DAQ Assistant, 11-2
- data
 - displaying
 - Graphical Array View, 4-10
 - displaying and editing
 - arrays, 4-7
 - strings, 4-8
 - variables, 4-5
 - generating random array of data, 6-3
 - reading with instrument driver, 8-4
- data acquisition
 - developing applications
 - example programs
 - NI-DAQ Library sample programs, 11-3
 - overview, 1-5, 11-1
 - software configuration, 11-2
 - testing operation and configuration, 11-2
- data acquisition boards
 - installation, software, 11-1
 - overview, 11-1
- data acquisition libraries, 1-3

- data analysis libraries, 1-3
- data analysis, programming overview, 1-6
- data files, functions for reading and writing, 9-4
- data presentation, libraries, 1-3
- Debug command, Run menu
 - running the completed project, 6-5
 - step mode execution, 4-2
- debugging programs
 - breakpoints
 - instant breakpoints, 4-3
 - predetermined breakpoints, 4-3
 - programmable breakpoints, 4-3
 - watch expression breakpoints, 4-3
 - displaying and editing data
 - Array Display, 4-7
 - String Display, 4-8
 - Variables window, 4-5
 - Watch window, 4-8
 - displaying data, Graphical Array View, 4-10
 - step mode execution, 4-1
- Debugging Region, 2-3
- Declare Variable command, Code menu
 - declaring arrays (example), 8-5
 - declaring variables (example), 7-4
- developing graphical user interfaces (GUI).
 - See* graphical user interface (GUI), building
- diagnostic tools (NI resources), A-1
- displaying and editing data
 - strings, 4-8
 - variables, 4-5
- documentation
 - conventions used in manual, *ix*
 - manuals for VXI controller boards, 10-4
 - NI resources, A-1
 - related documentation, *x*
 - related NI-DAQ documentation, 11-3
- documentation set, 1-1
- drivers (NI resources), A-1

E

Edit menu

- Source and Interactive Execution windows

- editing tools, 2-5

- Go To Definition command, 4-2

- Undo command, 2-6

- Workspace window

- Add Files to Project command, 8-8

editing data

- arrays, 4-8

- strings, 4-8

- tooltips, 4-9

- variables, 4-6

- editing tools in Source window, 2-5

- error messages, during compiling and linking, 2-4

- Error window. *See* Build Error window

- EVENT_COMMIT, 7-5

- EVENT_GOT_FOCUS, 7-5, 9-7

- EVENT_LEFT_CLICK, 7-5, 9-7

- EVENT_RIGHT_CLICK, 5-6, 9-7

events

- list of events, 9-7

- timed events, 9-8

- user interface events, 9-7

- example programs, NI-DAQ sample programs, 11-2, 11-3

- examples (NI resources), A-1

F

- file I/O functions, ANSI C library, 9-4

File menu

- Source and Interactive Execution windows, Open Quoted Text command, 2-5

- User Interface Editor, 5-3

finding functions, 3-7

- finding definitions, 4-2

- using CodeBuilder, 7-3

- Finish Function command, Run menu, 4-2

function panels

- accessing, 2-2, 3-2

- analyzing data, 3-7

- controls, 3-4

- declaring arrays, 6-3

- definition, 3-4

- executing functions interactively, 8-3

- finding functions, 3-7

- function definitions, 4-2

- using CodeBuilder, 7-3

- Generated Code pane, 3-6

- help, 3-4

- input control, 3-4

- inserting code from function panels, 3-6

- output control, 3-8

- purpose and use, 3-4

- recalling, 3-9

- selecting, 6-3

- testing operation of DAQ device and configuration, 11-2

- function trees (figure), 3-2

G

- General Purpose Interface Bus (GPIB). *See* GPIB boards

- Generate All Code command, Code menu, 5-6

- Generate All Code dialog box, 5-6

- Generate Control Callback command, 7-2

- Generated Code pane, 3-6

- generating code automatically. *See* CodeBuilder

- Go to Cursor command, Run menu, 4-2

- Go To Definition command, Edit menu, 4-2

GPIB boards

- compatible LabWindows/CVI interfaces, 10-2

- configuration
 - GPIB driver software, 10-2
 - LabWindows/CVI for GPIB, 10-2
- developing applications, 10-2
- installation, 10-2
- overview, 10-1
- graph controls, adding to user interface, 5-4
- Graphical Array View, displaying arrays, 4-10
- graphical user interface
 - building. *See* graphical user interface (GUI), building
 - example of GUI, 2-8
 - overview, 1-5
 - relationship of program elements (figure), 1-4
- graphical user interface (GUI), building
 - accessing the User Interface Library, 3-3
 - adding binary switch controls, 9-3
 - adding command button controls, 5-3
 - adding graph controls, 5-4
 - adding timer controls, 9-8
 - building a user interface resource (.uir) file, 5-2
 - constructing the project, 6-5
 - main function, 6-1
 - modifying, 7-1
 - running the program, 6-5, 7-5
 - saving .uir files, 5-5
 - setting up attributes
 - programmatically, 9-3
 - using instrument driver, 8-2
 - writing callback function, 7-3, 7-5
- graphs, drawing with function panels, 3-5
- GUI. *See* graphical user interface (GUI)

H

- header files, for user interface, viewing, 5-5
- help information, adding to controls, 9-8
- Help menu, Function Panel windows, 3-4

I

- IEEE 488 standard (GPIB), 10-1
- initializing instruments, 8-3
- input controls, adding to function panel, 3-4
- Insert Function Call command, Code menu
 - copying instrument driver code, 8-3, 8-4, 8-5
 - inserting code from function panel, 3-6
- installation
 - GPIB boards, 10-2
 - LabWindows/CVI components (table), 1-1
 - NI-DAQmx driver software, 11-1
 - VXI controllers, 10-3
- instant breakpoints, performing, 4-4
- instrument driver programming example, 8-1
 - adding sample program to project files, 8-8
 - closing the instrument drivers, 8-5
 - configuring instrument drivers, 8-3
 - functions in Sample Oscilloscope instrument, 8-2
 - initializing instrument drivers, 8-3
 - interactive function panel execution, 8-3
 - loading, 8-1
 - reading data, 8-4
 - reading waveform, 8-4
 - running the sample program, 8-6
 - Sample Oscilloscope program in Instruments folder, 8-2
- instrument drivers
 - available drivers, 10-4
 - definition, 10-4
- instrument drivers (NI resources), A-1
- Instruments folder
 - loading instrument drivers, 8-1
- interactive code generation tools
 - See also* CodeBuilder
 - analyzing data, 3-7
 - drawing graphs using function panels, 3-5

- executing a function panel
 - interactively, 3-10
- finishing the sample program, 3-9
- function panel controls, 3-4
- function panel help, 3-4
- inserting code from function panels, 3-6
- recalling function panels, 3-9

interprocess communication application
libraries, 1-3

K

KnowledgeBase, A-1

L

LabWindows/CVI

- See also* specific windows
- data acquisition, 11-1
- GPIB, 10-1
- learning about, 1-2
- libraries, 1-3
- location of components, 1-1
- overview, 1-1
- program development overview, 1-3
- starting, 2-1
- VXI Development System, 10-3

Library menu, Workspace window

- Customize Library command, 10-2, 10-4

Library Tree, 2-2, 3-1

- figure, 3-2

Line command, View menu, 2-6

Line Numbers command, View menu, 2-5

link error messages, 2-4

loading projects, 2-1

loading workspaces, 2-1

M

MAX, 10-4

Measurement & Automation Explorer, 10-4

N

National Instruments multimainframe
Resource Manager, 10-4

National Instruments support and
services, A-1

networking, libraries for, 1-3

Next Tag command, View menu, 2-6

NI support and services, A-1

NI-488.2 DLL, 10-2

NI-488.2 Library, 10-2

NI-DAQ driver software

- DAQ Assistant, 11-2
- MAX, 11-2
- NI-DAQ Library sample programs, 11-3

NI-DAQ Library, sample programs, 11-3

NI-Spy utility, 10-4

NI-VXI Library, 10-4

numeric controls

- adding to user interface, 7-3

O

Open command, File menu, 2-1

Open Quoted Text command, File menu, 2-5

Options menu, Workspace window

- Add Watch Expression command, 4-9
- Toolbar command, 2-3

Oscilloscope sample. *See* instrument driver
programming example

output controls, adding to function panel, 3-8

Output Region, 2-3, 2-4

P

pop-up panels, 9-5

Preview User Interface Header File command,
View menu, 5-5

program control

- components (figure), 1-4
- overview, 1-4

- program development overview
 - See also* source files
 - program structure for LabWindows/CVI
 - data acquisition, 1-5, 1-6
 - data analysis, 1-6
 - program control, 1-5
 - program shell generation with CodeBuilder, 1-5
 - relationship between elements (figure), 1-4
 - user interface, 1-5
 - using C in LabWindows/CVI, 1-4
- program shell, building. *See* CodeBuilder
- programming examples (NI resources), A-1
- programming graphical user interfaces. *See* graphical user interface (GUI), building
- programming tutorial
 - additional exercises, 9-1
 - adding channel control, 9-2
 - setting user interface attributes
 - programmatically, 9-3
 - storing data on disk, 9-4
 - timed events, 9-8
 - user interface events, 9-7
 - using pop-up panels, 9-5
 - debugging programs
 - breakpoints
 - manual breakpoints, 4-4
 - programmatic breakpoints, 4-3
 - displaying and editing data
 - Array Display, 4-7
 - String Display, 4-8
 - Variables window, 4-5
 - Watch window, 4-8
 - displaying data, Graphical Array View, 4-10
 - step mode execution, 4-1
 - editing tools, 2-5
- function panels
 - accessing, 2-2, 3-2
 - analyzing data, 3-7
 - controls, 3-4
 - declaring arrays, 6-3
 - drawing a graph, 3-5
 - executing functions interactively, 8-3
 - finding functions, 3-7
 - Generated Code pane, 3-6
 - help information, 3-4
 - input control, 3-4
 - inserting code from function panels, 3-6
 - output control, 3-8
 - recalling, 3-9
- graphical user interface
 - accessing User Interface Library, 3-3
 - adding binary switch controls, 9-3
 - adding command button controls, 5-3
 - adding graph controls, 5-4
 - adding timer controls, 9-8
 - analyzing source code, 6-1
 - building a user interface resource (.uir) file, 5-2
 - constructing projects, 6-5
 - example of GUI, 2-8
 - generating random array of data, 6-3
 - main function, 6-1
 - modifying, 7-1
 - running the program, 6-5, 7-5
 - saving .uir files, 5-5
 - setting attributes
 - programmatically, 9-3
 - source code connection, 5-1
 - writing callback function, 7-3

- instrument driver example
 - adding sample program to project files, 8-8
 - closing, 8-5
 - configuring, 8-3
 - initializing, 8-3
 - interactive function panel
 - execution, 8-3
 - interactive use, 8-2
 - loading, 8-1
 - reading data, 8-4
 - running the sample program, 8-6
- interactive code generation tools
 - accessing User Interface Library, 3-3
 - executing function panels
 - interactively, 3-10
 - finishing the sample program, 3-9
 - function panel fundamentals, 3-4
- Library Tree, 3-1
- loading projects, 2-1
- running projects, 2-3
- Project Tree, 2-2
- projects
 - See also* source files
 - accessing and viewing files within, 2-2
 - adding files to projects, 8-8
 - adding instruments, 8-8
 - error messages, 2-4
 - loading, 2-1

R

- random array of data, generating, 6-3
- reading data with instrument driver, 8-4
- reading waveform (example), 8-4
- Recall Function Panel command,
 - View menu, 3-9
- related documentation, 11-3
- Run Function Panel command,
 - Code menu, 8-3, 8-4, 8-5

Run menu

- Source and Interactive Execution windows
 - Continue command, 4-2, 4-9
 - Debug command, 4-1, 6-5
 - Step Into command, 4-2, 4-4, 4-6
 - Step Over command, 4-2, 4-4
 - Terminate Execution command, 4-2, 4-4
 - View Variable Value command, 4-7
- Workspace window
 - Break on First Statement command, 4-1, 4-5, 4-8
 - Debug command, 6-5

S

- Sample Oscilloscope program. *See* instrument driver programming example
- sample programs. *See* example programs
- Set Next Statement command, Run menu, 4-2
- Set Target File command, Code menu, 3-6
- shells, building. *See* CodeBuilder
- software (NI resources), A-1
- solutions to tutorial, 9-1
- source files
 - analyzing code, 6-1
 - displaying in Source window, 2-4
 - displaying referenced files, 2-5
 - error messages, 2-4
 - inserting code from function panels, 3-7
 - loading, 2-1
 - moving to specific lines of code, 2-6
 - opening subwindows for one source file, 2-5
 - recalling function panel, 3-9
 - source code connection, 5-1

Source window

- compatibility with ANSI C specifications, 2-4
- displaying generated code, 5-6
- displaying source code, 2-4
- editing tools, 2-5
- opening subwindows, 2-5

Standard Input/Output Window, 2-4

starting LabWindows/CVI, 2-1

Step Into command, Run menu, 4-1, 4-2, 4-4, 4-6

step mode execution, 4-1

Step Over command, Run menu, 4-2, 4-4

String Display window, displaying and editing string variables, 4-8

subwindows

- illustration, 2-6
- opening subwindows for one source file, 2-5

support, technical, A-1

T

tagged lines, 2-6

technical support, A-1

Terminate Execution command, Run menu, 4-2, 4-4

testing board configuration, 11-2

timed events, 9-8

timer controls, adding to user interface, 9-8

Toggle Tag command, View menu, 2-6

Toolbar command, Options menu, 2-3

toolbars

- customizing, 2-3
- displaying names of buttons or icons, 2-3

tooltips, 4-9

- editing variables, 4-9

training and certification (NI resources), A-1

troubleshooting (NI resources), A-1

U

uir files. *See* user interface resource (.uir) files

Undo command, Edit menu, 2-6

User Interface Editor window, purpose and use, 5-1

user interface events. *See* events

User Interface Library

- accessing, 3-3
- contents, 1-5

user interface resource (.uir) files,

- building, 5-2
 - adding a graph control, 5-4
 - adding command button controls, 5-3

user interface, creating. *See* graphical user interface (GUI), building

user interface. *See* graphical user interface (GUI)

V

variables

- displaying, 4-5
- editing, 4-6
- editing using tooltips, 4-9

Variables window

- opening, 4-5
- stepping through programs, 4-5

View Control Callback command, 7-3

View menu

- Line command, 2-6
- Line Numbers command, 2-5
- Next Tag command, 2-6
- Preview User Interface Header File, 5-5
- Toggle Tag command, 2-6

View Variable Value command, Run menu, 4-7

VISAIC, 10-4

- VXI controllers
 - compatible controllers, 10-3
 - configuration
 - configuring LabWindows/CVI for VXI, 10-4
 - VXI driver software, 10-4
 - VXI hardware, 10-3
 - developing applications, 10-4
 - documentation for VXI controller boards, 10-4
 - installation, 10-3
 - overview, 10-3

W

- watch expressions, 4-3
- Watch window
 - Add/Edit Watch Expression dialog box, 4-9
 - displaying variables during program execution, 4-8
 - purpose and use, 4-8

- waveform generation project
 - See also* instrument driver programming example; user interface (figure)
 - adding a channel control, 9-2
 - reading waveform, 8-5
 - storing waveform on disk, 9-4
- Web resources, A-1
- Window Confinement Region, 2-2
- Window menu, Project window, Build Errors command, 2-4
- windows, managing windows (note), 2-1
- Workspace window, 2-2
- workspaces, loading, 2-1