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.

APEX WAVES

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 *PCMCIA-GPIB*

LabWindows/CVI

Getting Started with LabWindows/CVI

NATIONAL INSTRUMENTS™

December 1999 Edition
Part Number 320680E-01

Worldwide Technical Support and Product Information

www.ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00, Poland 48 22 528 94 06,
Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

© Copyright 1994, 1999 National Instruments Corporation. All rights reserved.

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 THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

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

Trademarks

CodeBuider™, CVI™, DataSocket™, National Instruments™, ni.com™, NI-DAQ™, NI-488.2™, and NI-488.2M™ are trademarks of National Instruments Corporation.

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

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 xiii
Related Documentation xiii

Chapter 1
Introduction to LabWindows/CVI

Installing LabWindows/CVI 1-1
How to Learn About LabWindows/CVI Quickly 1-2
LabWindows/CVI System Overview 1-2
LabWindows/CVI Program Development Overview 1-4
Using C in LabWindows/CVI 1-4
LabWindows/CVI Program Structure 1-4
User Interface 1-5
Program Shell Generation with CodeBuilder 1-5
Program Control 1-5
Data Acquisition 1-6
Data Analysis 1-7

Part I
Getting Acquainted with the LabWindows/CVI Development Environment

Chapter 2
Loading, Running, and Editing Source Code

Note about LabWindows/CVI Windows 2-1
Setting Up 2-2
Loading a Project into LabWindows/CVI 2-2
Project Window 2-4
Running the Project 2-6
Error Messages 2-6
Standard Input/Output Window 2-6
Source Window 2-6
Editing Tools 2-8
Operating Projects with a User Interface 2-10

Chapter 3
Interactive Code Generation Tools

Setting Up 3-1
Library Menu 3-1
Accessing the User Interface Library 3-3
Function Panel Fundamentals 3-4
Function Panel Controls 3-5
Function Panel Help 3-5
Drawing a Graph 3-5
Inserting Code from a Function Panel 3-7
Analyzing Data 3-8
Output Values on a Function Panel 3-9
Recalling a Function Panel 3-10
Finishing the Program 3-10
Interactively Executing a Function Panel 3-12

Chapter 4
Executing and Debugging Tools

Setting Up 4-1
Step Mode Execution 4-2
Breakpoints 4-4
Programmatic Breakpoints 4-4
Manual Breakpoints 4-6
Displaying and Editing Data 4-6
Variables Window 4-7
Editing Variables 4-9
Array Display 4-11
Editing Arrays 4-12
String Display 4-13
Watch Window 4-13

Part II
Building an Application in LabWindows/CVI

Chapter 5
Building a Graphical User Interface

Setting Up 5-1
User Interface Editor 5-2
Source Code Connection 5-2
CodeBuilder 5-3

Sample Project 5-3
Building a User Interface Resource (.uir) File 5-3
Opening a .uir File 5-4
Adding a Command Button 5-5
Adding a Graph Control to the User Interface 5-7
Saving the .uir File 5-9
Generating the Program Shell with CodeBuilder 5-10

Chapter 6
Using Function Panels and the Libraries

Setting Up 6-1
Analyzing the Source Code 6-1
main Function 6-2
AcquireData Function 6-2
Shutdown Function 6-3
Generating a Random Array of Data 6-3
Finding the PlotY Function 6-4
Building the PlotY Function Call Syntax 6-8
Constructing the Project 6-10
Running the Completed Project 6-12

Chapter 7
Adding Analysis to Your Program

Setting Up 7-1
Goals of Section 7-2
Modifying the User Interface 7-2
Writing the Callback Function 7-6
Running the Program 7-9

Chapter 8
Using an Instrument Driver

Setting Up 8-1
Loading the Instrument Driver 8-1
Using the Instrument Driver 8-3
Interactive Function Panel Execution 8-4
Initializing the Instrument 8-4
Configuring the Instrument 8-6
Reading Data with an Instrument Driver 8-7
Declaring Arrays from Function Panels 8-8
Reading the Waveform 8-9
Closing the Instrument 8-10

Running the Program 8-10
Adding the Instrument to Your Project 8-12

Chapter 9
Additional Exercises

Base Project 9-1
Exercise 1: Adding a Channel Control 9-2
Assignment 9-2
Hints 9-3
Exercise 2: Setting User Interface Attributes Programmatically 9-4
Assignment 9-4
Hints 9-4
Exercise 3: Storing the Waveform on Disk 9-5
Assignment 9-5
Hints 9-5
Exercise 4: Using Pop-up Panels 9-6
Assignment 9-6
Hints 9-7
Exercise 5: User Interface Events 9-8
Assignment 9-9
Hints 9-9
Exercise 6: Timed Events 9-10
Assignment 9-10
Hints 9-10

Part III
Instrument Control, Data Acquisition, and LabWindows for DOS Conversions

Chapter 10
Getting Started with GPIB and VXI Instrument Control

Getting Started with Your GPIB Controller 10-1
Introduction to GPIB 10-1
Installing Your GPIB Interface Board 10-2
Configuring Your GPIB Driver Software 10-2
Configuring LabWindows/CVI for GPIB 10-3
Developing Your Application 10-3
Getting Started with Your VXI Controller 10-4
Introduction to VXI 10-4
VXI Development System 10-4
Installing and Configuring Your VXI Hardware 10-4

Configuring Your VXI Driver Software 10-5
Configuring LabWindows/CVI for VXI 10-5
Developing Your Application 10-6
Using Instrument Drivers 10-7

Chapter 11
Getting Started with Data Acquisition

Introduction to Data Acquisition 11-1
Installing Your DAQ Device 11-2
Configure Your Jumpers and DIP Switches 11-2
Software Installation 11-2
Configuring LabWindows/CVI for Data Acquisition 11-3
Test the Operation of Your Device and Configuration 11-4
Develop Your Application 11-4
Easy I/O for DAQ Library Sample Programs 11-5
Data Acquisition Library Sample Programs 11-5
DAQ Control Instrument Drivers 11-5
DAQ Numeric Control Instrument Driver 11-5
DAQ Chart Control Instrument Driver 11-5
Event Function Parameter Data Types 11-6
Source Code Changes Needed 11-7
Related Documentation 11-7

Chapter 12
Converting LabWindows for DOS Applications

Conversion Tools 12-1
Unsupported Features 12-2
Functions with New Behaviors 12-3
User Interface Library 12-3
Utility Library 12-3
Converting User Interface Resource (.uir) Files 12-3
Converting Source Code 12-4
Converting Instrument Drivers 12-7
Convert the Instrument Driver Function Panels 12-7
Convert the Instrument Driver Header File 12-8
Convert the Instrument Driver Source Code 12-8
Converting Loadable Compiled Modules and External Modules 12-8

Appendix A
Technical Support Resources

Glossary

Index

Figures

Figure 1-1. Relationship of Program Elements in LabWindows/CVI 1-5

Figure 2-1. Project Window 2-1
Figure 2-2. Source Window 2-2
Figure 2-3. User Interface Editor Window 2-2
Figure 2-4. Open File Dialog Box 2-3
Figure 2-5. sample1.prj in the Project Window 2-3
Figure 2-6. Adding a File to a Project 2-5
Figure 2-7. Information in the Project Window 2-5
Figure 2-8. Source Code for the sample1.c File 2-7
Figure 2-9. Split Source Window 2-9
Figure 2-10. sample2.prj Files in the Project Window 2-10
Figure 2-11. sample2.uir Panel When Running 2-10

Figure 3-1. Library Menu 3-2
Figure 3-2. User Interface Library Function Tree 3-2
Figure 3-3. Y Graph Popup Function Panel 3-4
Figure 3-4. Completed Y Graph Popup Function Panel 3-6
Figure 3-5. Source Window with Code Inserted from Function Panel 3-7
Figure 3-6. Find Function Panel Dialog Box 3-8
Figure 3-7. Completed Function Panel for Mean 3-9
Figure 3-8. Replace/Insert Dialog Box 3-10

Figure 4-1. sample3.c Program 4-2
Figure 4-2. Breakpoint beside a Line of Code 4-5
Figure 4-3. Variables Window 4-7
Figure 4-4. Variables Window during Execution of main 4-8
Figure 4-5. Variables Window during Execution of get_and_print_random 4-9
Figure 4-6. Variables Window with i Variable Highlighted 4-10
Figure 4-7. Edit Value Dialog Box 4-10
Figure 4-8. Array Display 4-11
Figure 4-9. Edit Value Dialog Box 4-12
Figure 4-10. String Display 4-13
Figure 4-11. Add/Edit Watch Expression Dialog Box 4-14

Figure 5-1. Transfer Project Options Dialog Box 5-2
Figure 5-2. User Interface Editor with GUI Panel 5-4
Figure 5-3. User Interface Editor with Completed GUI Panel 5-5
Figure 5-4. Edit Command Button Dialog Box 5-6
Figure 5-5. Source Code Connection Section of Edit Command Button Dialog Box 5-7
Figure 5-6. User Interface with Two Buttons 5-7
Figure 5-7. Edit Graph Dialog Box 5-8
Figure 5-8. User Interface Header File 5-9
Figure 5-9. Control Callback Events Dialog Box 5-10
Figure 5-10. Generate All Code Dialog Box 5-11

Figure 6-1. For Loop Dialog Box 6-4
Figure 6-2. Select Function Panel Dialog Box 6-5
Figure 6-3. Graph Plotting and Deleting Functions 6-6
Figure 6-4. Plot Y Function Panel 6-7
Figure 6-5. Select UIR Constant Dialog Box 6-8
Figure 6-6. Completed Plot Y Function Panel 6-9
Figure 6-7. Completed Source Code for sample4.c 6-10
Figure 6-8. Add Files to Project Dialog Box 6-11

Figure 7-1. Sample User Interface 7-3
Figure 7-2. CodeBuilder—Generated Code 7-4
Figure 7-3. Selecting a Numeric Control from the Create Menu 7-5
Figure 7-4. Using Code Builder to Find and Highlight the FindMaxMin Function 7-6
Figure 7-5. Completed Source Code for sample5.c 7-8

Figure 8-1. sample6.c Source File with Tags 8-2
Figure 8-2. Sample Oscilloscope on Instrument Menu 8-2
Figure 8-3. Sample Oscilloscope Functions 8-3
Figure 8-4. Initialize Function Panel 8-4
Figure 8-5. Configure Function Panel 8-6
Figure 8-6. Read Waveform Function Panel 8-8
Figure 8-7. Declare Variable Dialog Box 8-9

Figure 9-1. Sample User Interface 9-2
Figure 9-2. User Interface with Channel Selection Control 9-3
Figure 9-3. File Select Popup 9-6
Figure 9-4. Completed User Interface 9-7

Figure 10-1. Library Options Dialog Box with GPIB/GPIB 488.2 Selected 10-3
Figure 10-2. Library Options Dialog Box with VXI Selected 10-6

Figure 11-1. Library Options Dialog Box with Easy I/O for DAQ Selected 11-3

Figure 12-1. Select System Font Dialog Box 12-4
Figure 12-2. Translate LW DOS Program Dialog Box 12-5

Tables

Table 1-1. LabWindows/CVI Components 1-1

Table 2-1. File Types and the Corresponding Windows 2-1
Table 2-2. Project Window Menus 2-4

Table 3-1. Panel Help Display Procedures 3-5

Table 4-1. Quick Keys for Step Mode Execution 4-3

Table 11-1. Typedefs and Intrinsic Types for Different Platforms 11-6

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, Windows, and the C programming language.

Conventions

The following conventions appear in this manual:

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

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

bold Bold text denotes items that you must select or click on 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, and code excerpts.

Related Documentation

The following document contains information that you may 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.

1

Introduction to LabWindows/CVI

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

Installing LabWindows/CVI

The LabWindows/CVI installation includes sample programs that illustrate many of the new features in LabWindows/CVI and tutorial programs that you use throughout this manual. 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
`c:\cvi\bin`	LabWindows/CVI Library files and online help files
`c:\cvi\extlib`	Files for using the LabWindows/CVI libraries with external compilers
`c:\cvi\fonts`	Font files required for graphics operations
`c:\cvi\include`	Include files associated with libraries
`c:\cvi\instr`	Instrument modules
`c:\cvi\samples`	Source code to sample programs
`c:\cvi\sdk`	Software Developer's Kit (SDK) library files
`c:\cvi\toolslib`	Additional development tools and libraries
`c:\cvi\tutorial`	Programs used in the tutorial exercises throughout this manual
`c:\cvi\vxd`	`vxd` sample code templates

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

How to Learn About LabWindows/CVI Quickly

You can learn about LabWindows/CVI as follows:

1. Read the remainder of this chapter to learn about the concepts and capabilities of LabWindows/CVI.
2. Complete the tutorial exercises in Chapters 2–9 as described in this manual. Each section of the tutorial builds on previous elements. Therefore, you should follow the outline as given and not skip ahead.
3. Familiarize yourself with the sample programs that come with LabWindows/CVI.

Chapter 2, *Loading, Running, and Editing Source Code*, introduces the windows, menus, commands, and dialog boxes that you use in LabWindows/CVI. The *LabWindows/CVI User Manual* gives detailed information on these topics. For example, when the tutorial instructs you to use the Variable display window or function panels, you can refer to the *LabWindows/CVI User Manual* for detailed information on these topics.

As you work through the tutorial, you can refer to the online help to see reference information for the function libraries. Take time to acquaint yourself with each of the sections in the *LabWindows/CVI Online Help*. Study the function tree of each library that you will use in your own development work. The function trees show the categories within each LabWindows/CVI library and give you a longer, descriptive name with each function name.

Beginners should complete the tutorial exercises in this manual first. Then they are ready to read the *LabWindows/CVI User Manual*. These two manuals contain the fundamental information that users require. For a listing of documentation and the location of specific LabWindows/CVI information, refer to the *About This Manual* section of the *LabWindows/CVI User Manual*.

LabWindows/CVI System Overview

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

- Interactively develop programs
- Access powerful libraries of functions 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. In the environment, you use the functions in the LabWindows/CVI function libraries to write your program. In addition, each function has an interface called a

function panel where you can execute the function and generate code for calling the function. While you work with function panels, you can consult online help for the function itself and for each control on the function panel.

Programs written in the LabWindows/CVI interactive environment must adhere to the ANSI C specification. In addition, you are free to use compiled C object modules, dynamic link libraries (DLLs), C libraries, and instrument drivers in conjunction with ANSI C source files when you develop your programs. Refer to the *LabWindows/CVI Programmer Reference Manual* for information on LabWindows/CVI loadable object modules and DLLs.

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

- **Data Acquisition, Seven Libraries**—Instrument Library, GPIB/GPIB 488.2 Library, Data Acquisition Library, Easy I/O for DAQ, RS-232 Library, VISA Library, VXI Library
- **Data Analysis, Three Libraries**—Formatting and I/O Library, Analysis Library, optional Advanced Analysis Library
- **Data Presentation**—User Interface Library
- **Networking and Interprocess Communication Applications, Four Libraries**—Dynamic Data Exchange (DDE) Library, Transmission Control Protocol (TCP) Library, ActiveX Automation Library, DataSocket Library

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

The Instrument Library is a special resource in LabWindows/CVI. It contains drivers for GPIB, VXI, and RS-232 instruments such as oscilloscopes, multimeters, and function generators. Each driver comes with source code so that you can modify it if necessary.

LabWindows/CVI development tools permit you to create your own instrument drivers. You can create instrument drivers for a single instrument, multiple instruments, or a virtual instrument for which no physical instrument exists. You use functions from the other LabWindows/CVI libraries to create instrument drivers.

The User Interface Library contains tools that you use to control graphical user interfaces (GUIs) from your application programs. LabWindows/CVI has a User Interface Editor where you create GUIs. You use functions in the User Interface Library to control your GUI. You can control GUI panels that contain *input* and *output controls*, graphs, and strip charts. You can also 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 all these items. Alternatively, you can use the User Interface Library to create these items programmatically.

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 do 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.

Using C in LabWindows/CVI

LabWindows/CVI enhances the C programming language for instrumentation applications. Before you begin working with LabWindows/CVI, you should have a fundamental understanding of C programming. For a description of the ANSI C Standard Library as implemented in LabWindows/CVI, refer to the topics in the ANSI C Library section of the *LabWindows/CVI Online Help*.

To learn more about the enhancements to ANSI C in LabWindows/CVI, you can examine the sample program listings located in the `cvi55\samples directory`. These sample programs illustrate LabWindows/CVI User Interface features and show how to use the LabWindows/CVI library functions.

LabWindows/CVI Program Structure

Because LabWindows/CVI is flexible, you can take almost any approach to building a program. Most of the projects you build probably include the following elements:

- User interface
- Program control
- Data acquisition
- Data analysis

Figure 1-1 shows a rough outline of program elements. Program Control elements receive input from the User Interface, Data Acquisition, and Data Analysis elements. Each element has several sub-components.

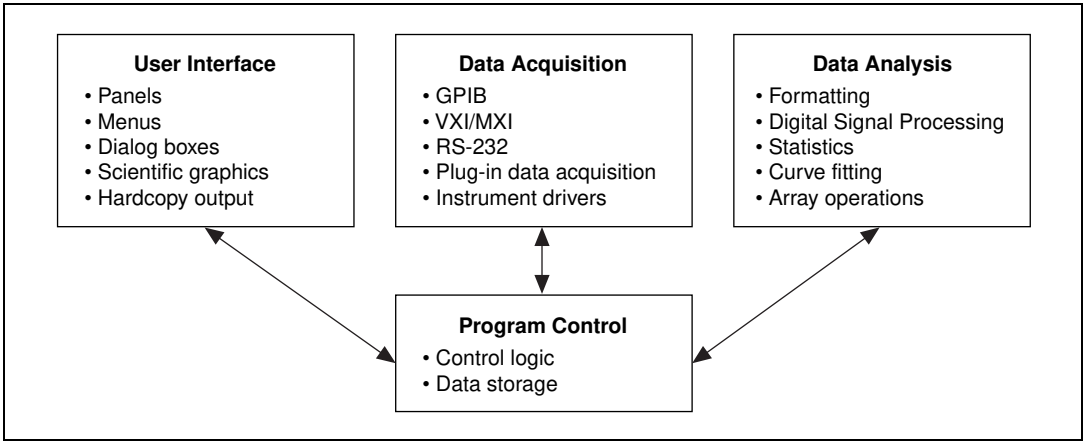

Figure 1-1. Relationship of Program Elements in LabWindows/CVI

User Interface

With the LabWindows/CVI User Interface Editor you can build complex, interactive panels for your program with minimum effort. During the process of GUI design, developers often decide exactly how they want their program 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 your program.

To learn more about the elements of the user interface and the functions that allow you to connect your interface to the rest of your program, refer to the *Graphical User Interface* section of the *LabWindows/CVI Online Help*.

Program Shell Generation with CodeBuilder

After you design your GUI in the User Interface Editor, you can use CodeBuilder to automatically generate a program shell based on the components in your GUI. CodeBuilder writes code for all your control callback functions and creates a main program to load and display your GUI windows at program startup. CodeBuilder saves hours of 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 your program coordinates data acquisition, data analysis, and the user interface. It 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 sample programs that come with LabWindows/CVI to better understand how to write code for controlling your program. Pay attention to the use of callback functions in the sample programs. Callback functions can help you control the flow of your application easily.

Data Acquisition

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

The various LabWindows/CVI libraries provide functions for creating the data acquisition program element. Functions exist for controlling GPIB, RS-232, and VXI devices and National Instruments DAQ devices. In addition, the LabWindows/CVI Instrument Library contains a collection of drivers for many popular GPIB, RS-232, and VXI instruments.

GPIB functions are introduced in the GPIB Library topics of the *LabWindows/CVI Online Help*, with detailed function descriptions available in the NI-488.2 online help that comes with your GPIB interface. VXI library functions are documented in the *NI-VXI Programmer Reference Manual* that comes with your VXI controller.

The *NI-DAQ Function Reference Manual for PC Compatibles* and the *NI-DAQ Function Reference Online Help*, distributed with National Instruments DAQ devices, document the Data Acquisition library functions.This documentation contains a table that describes which functions apply to specific DAQ devices along with specific information on using those functions in LabWindows/CVI.

The functions in the Easy I/O for DAQ Library make it easier to write simple DAQ programs than if you use the Data Acquisition Library. This library implements a subset of the functionality of the Data Acquisition Library, but it does not use the same functions as the Data Acquisition Library. For more information, refer to the function panel help for the Easy I/O for DAQ Library as follows:

- Select **Library»Easy I/O for DAQ** in LabWindows/CVI.
- In the Select Function Panel dialog box that appears, click the function that you want to learn about and click **Select**.
- In the Function Panel window that appears, you can right-click to access help for controls, for the function itself, and for the class grouping of the function.

In the LabWindows/CVI Instrument Library you can choose from a variety of drivers for GPIB, RS-232, and VXI instruments. Refer to the *Using Instrument Drivers* and the *Instrument Menu* sections of Chapter 3, *Project Menu*, in the *LabWindows/CVI User Manual*.

Also, Chapter 10, *Getting Started with GPIB and VXI Instrument Control*, and Chapter 11, *Getting Started with Data Acquisition*, in this manual provide further detail regarding these topics.

Data Analysis

After you acquire data, it is often necessary to analyze it. For example, you might want to perform formatting, scaling, signal processing, statistics, and curve fitting. The following function libraries contain functions that perform these operations:

- Formatting and I/O Library
- Analysis Library
- Advanced Analysis Library (optional)

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

Part I

Getting Acquainted with the LabWindows/CVI Development Environment

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

2

Loading, Running, and Editing Source Code

In this chapter, you load and run various projects in the LabWindows/CVI development environment, and you 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

Note about LabWindows/CVI Windows

In this tutorial you use three windows of the LabWindows/CVI development environment: the Project window, the Source window, and the User Interface Editor window. Unlike other windows in LabWindows/CVI, the title bar of these windows displays the name of the file you are working with. The file extension is your clue to which window you are in. Table 2-1 shows file types and the corresponding LabWindows/CVI windows:

Table 2-1. File Types and the Corresponding Windows

When You Open This File Type:	You See This Window:
`*.prj`	Project window
`*.c`	Source window
`*.uir`	User Interface Editor window

The Figures 2-1, 2-2 and 2-3 show the top portion of these window types for the `sample1.prj` sample project.

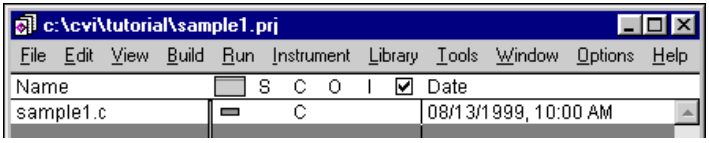

Figure 2-1. Project Window

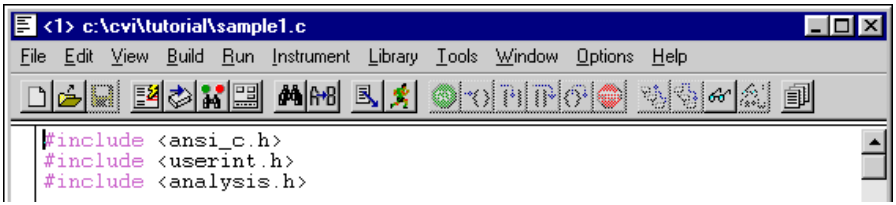

Figure 2-2. Source Window

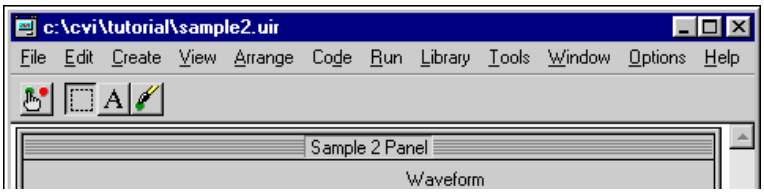

Figure 2-3. User Interface Editor Window

Setting Up

Launch LabWindows/CVI by selecting **Start»Programs»National Instruments CVI»CVI IDE**. When LabWindows/CVI opens, you see an empty Project window.

Note Instead of using LabWindows/CVI commands to manipulate windows—close, maximize, minimize, position, and more—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.

Loading a Project into LabWindows/CVI

To view some of the editing and execution features of the LabWindows/CVI development environment, load a project into the LabWindows/CVI Project window as follows:

1. Select **File»Open**. A submenu appears, showing the different file types that you can create and edit in LabWindows/CVI.
2. Select **Project(*.prj)** to invoke the Open File dialog box shown in Figure 2-4. You might need to open the `cvi` directory, then open the `tutorial` directory to access the sample project files.

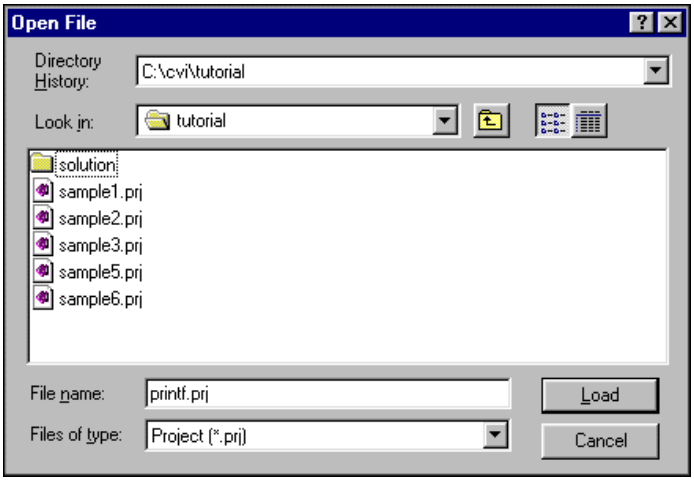

Figure 2-4. Open File Dialog Box

3. Select the `sample1.prj` project in the `tutorial` subdirectory.

After you load `sample1.prj`, the Project window appears as shown in Figure 2-5.

Note If a previous user has altered the sample files for this tutorial, reinstall LabWindows/CVI to get the original versions. You can perform a custom installation that installs only the tutorial files.

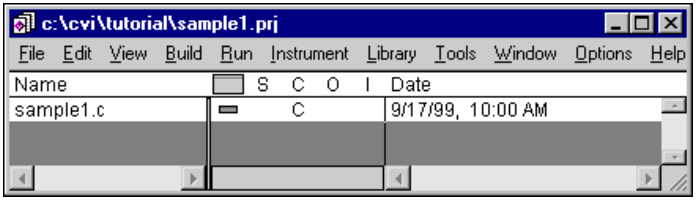

Figure 2-5. sample1.prj in the Project Window

The next section presents several more of the windows available in LabWindows/CVI: the Project window, the Standard Input/Output window, and the Source window.

Project Window

The Project window in LabWindows/CVI lists all the files that make up a particular project or program. The Project window contains the menus as shown in Table 2-2.

Table 2-2. Project Window Menus

Menu	Options
File	Save and create files. Open files types for a LabWindows/CVI project: project (`.prj`), source (`.c`), header (`.h`), or user interface (`.uir`).
Edit	Add or remove files from the project list.
Build	Use the compiler and linker in LabWindows/CVI.
Run	Run a project.
Window	Access open windows in LabWindows/CVI quickly, such as the Source window, User Interface Editor window, and Standard Input/Output window.
Tools	Run wizards and any executables that you add to the **Tools** menu.
Options	Configure various aspects of the LabWindows/CVI programming environment.
Help	Access help for LabWindows/CVI and for Windows SDK functions

This manual introduces you to many windows, utilities, and editors. For more information on the Project window menus, refer to Chapter 3, *Project Window*, of the *LabWindows/CVI User Manual*.

You learn about the **Instrument** and **Library** menus in subsequent chapters of this tutorial.

Note To add many of the pre-existing files to your project—C source files, header files, object modules, DLL, C libraries, user interface files, and instrument drivers—select **Edit»Add File to Project** and select the type of file that you want to add to your project, as shown in Figure 2-6.

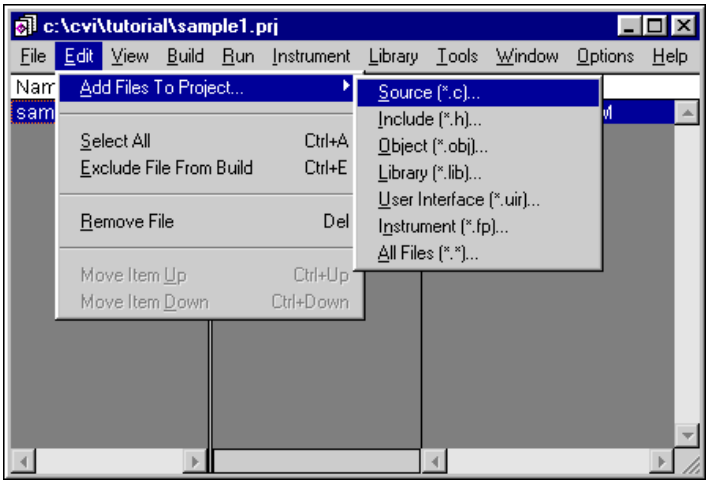

Figure 2-6. Adding a File to a Project

The simple project you use in this part of the tutorial, `sample1.prj`, contains only a single C source file. The Project window displays status information for the files listed in the project list. Figure 2-7 shows how this information appears in the Project window.

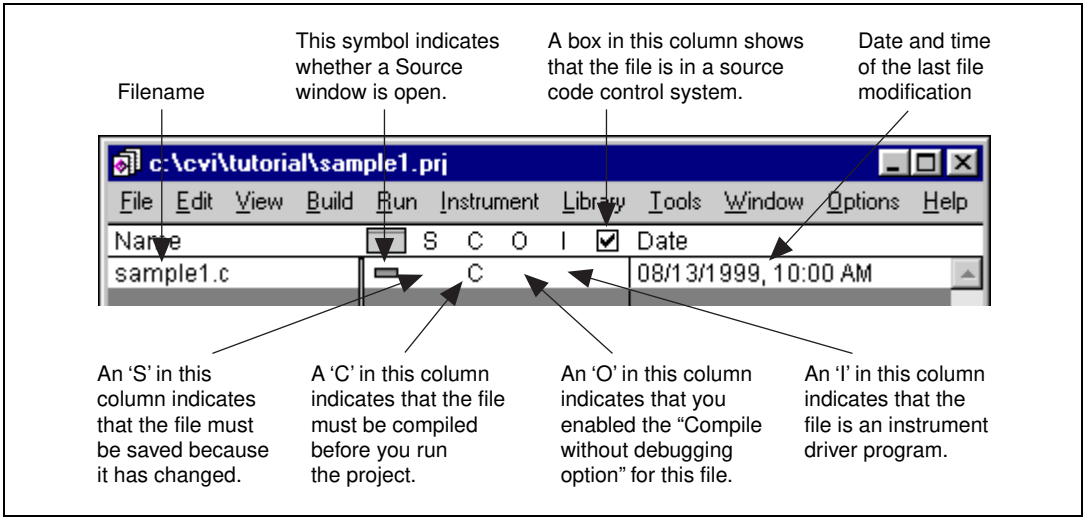

Figure 2-7. Information in the Project Window

Running the Project

To run the `sample1` project, select **Run»Debug sample1_dbg.exe**. LabWindows/CVI responds to the Run Project command by:

- Compiling any source files in the project list
- Linking the project with the libraries used
- Executing the compiled code
- Turning off the C indicator in the project list after the source compiles
- Displaying the word "Running" in the upper left-hand corner of the Project window while the project runs

The `sample1` project is a simple 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, an error message appears. The error message window contains the number of errors LabWindows/CVI detects in each source file and a description of the current error. For example, you can get an illegal character error or a syntax error in which case the Build Errors window appears at the bottom of your screen. The type of error you have is highlighted, and the line number of the error appears to the left of the highlighted error type. Correct the error and rerun your program.

To close the error message window, click the **Close** box located in the upper left-hand corner of the window. Select **Window»Build Errors** or **Window»Runtime Errors** to make the Error window or the Runtime Errors window appear.

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

The Source window in LabWindows/CVI is where you develop C source files for your projects. After running the `sample1` project, complete the following steps to view some of the features in the Source window in LabWindows/CVI.

1. Close the Standard Input/Output window by pressing <Enter> in the Standard Input/Output window.
2. Double-click the `sample1.c` filename in the Project window to display the source code in a Source window.

3. The source code appears as shown in Figure 2-8. As you can see, the source code for `sample1` contains standard ANSI C compatible code.

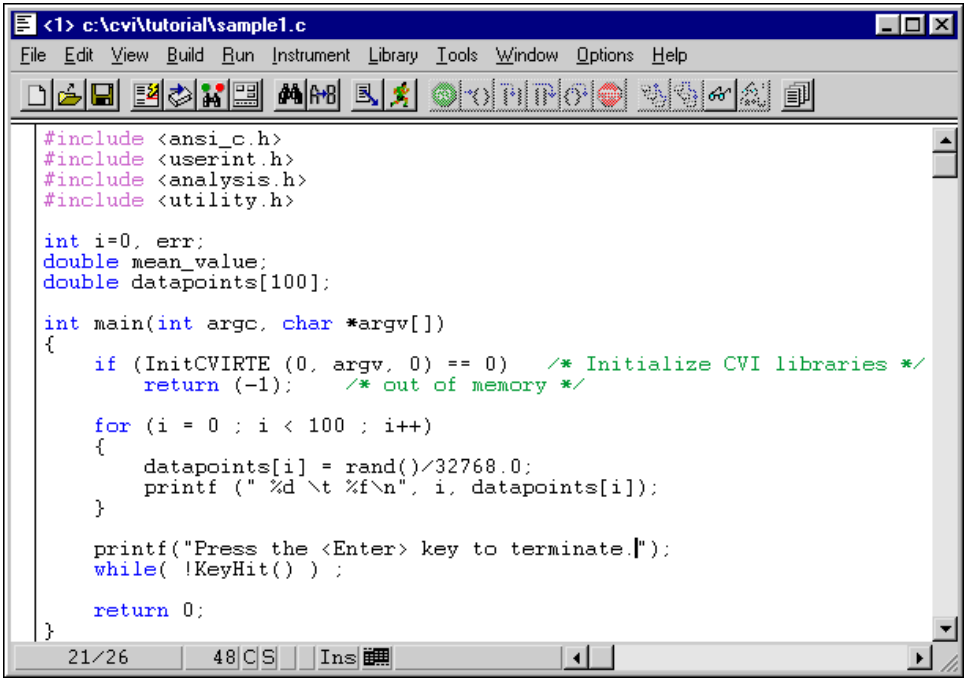

Figure 2-8. Source Code for the sample1.c File

The **File** menu in the Source window is very similar to the **File** menu in the Project window. Under the **File** menu, you can open, save, or create any type of file on which LabWindows/CVI can operate.

The **Edit** menu contains source code editing selections.

The **Run** menu contains selections for debugging your source code during run time.

The **Instrument** menu accesses any instrument drivers loaded in the system.

The **Library** menu accesses the LabWindows/CVI libraries for performing data acquisition, analysis, and presentation operations.

The Source window menus are described more fully in Chapter 5, *Source and Interactive Execution Windows*, of the *LabWindows/CVI User Manual*.

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 learn more about code generation tools in later chapters of this tutorial.

The Source window, Function Panel windows, and Variables displays 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, you can place your mouse cursor over the icon and the built-in tooltips appear to show which menu command the icon represents. You can customize the icons in your toolbar from the by selecting **Options»Toolbar**.

Editing Tools

The LabWindows/CVI Source window has a number of quick editing features that are helpful when working with large source files or projects with a large number of source files. The following exercise illustrates some of these editing features. Most of the features described here are located in the **Edit** menu. The **Edit** menu contains standard Windows editing features, such as **Cut**, **Copy**, **Paste**, **Find**, and **Replace**. The arrow positioning keys, <Page Up>, <Page Down>, <Home>, and <End>, operate in the Source window in a fashion similar to a word processor.

1. If you are viewing a large file, select **View»Line Numbers** to refer to particular line numbers. A new column appears to the left of the window with line numbers displayed.
2. Many times, the programs you develop in LabWindows/CVI refer to other files, such as header files or user interface files. You can view these additional files quickly by placing the cursor on the filename in your source code and selecting **File»Open Quoted Text**, or by pressing <Ctrl-u>.

   Place the cursor on the `ansi_c.h` filename on line 1 of `sample1.c` and press <Ctrl-u>. The `ansi_c` header file is displayed in a separate Source window. Scroll through the `ansi_c.h` 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 by selecting **File»Close**. The `sample1.c` file becomes in the active window.
3. If you are working on a large source file and must view a portion of your source code while you are making 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. You see a duplicate copy of the source code in each subwindow as shown in Figure 2-9.

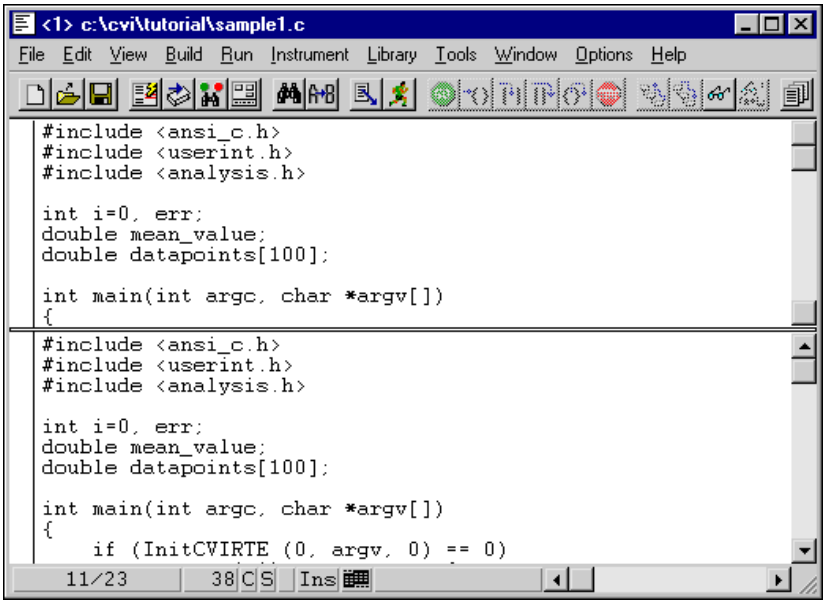

Figure 2-9. Split Source Window

4. 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 enter some text from the keyboard. Notice that the text appears in both halves of the window.
5. 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.

   Select **Edit»Undo**. The text you entered in Step 4 on line 5 of the source code disappears.
6. Drag the dividing line between the two subwindows back to the top of the source window to make a single window again.
7. Two different methods exist in LabWindows/CVI for quickly moving 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.

   Setting tags on particular lines is another method 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.
8. Move the cursor to line 12 of the Source window and enter 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.

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. The purpose of this section is to briefly show you how a GUI looks and works. Complete the following steps to load a new sample program.

1. Select **File»Open** and choose **Project (*.prj)** as the file type. Choose `sample2.prj` from the Open File dialog box. The Project window appears as shown in Figure 2-10.

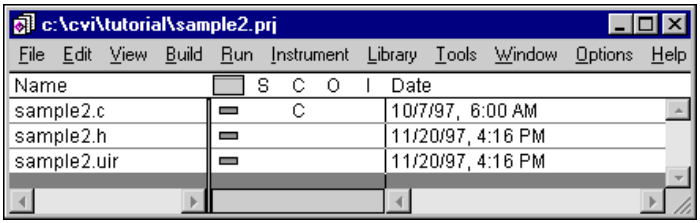

Figure 2-10. sample2.prj Files in the Project Window

2. Run the project by selecting **Run»Debug sample2_dbg.exe**, or by pressing <Shift-F5>. The GUI shown in Figure 2-11 appears after the program compiles and runs.

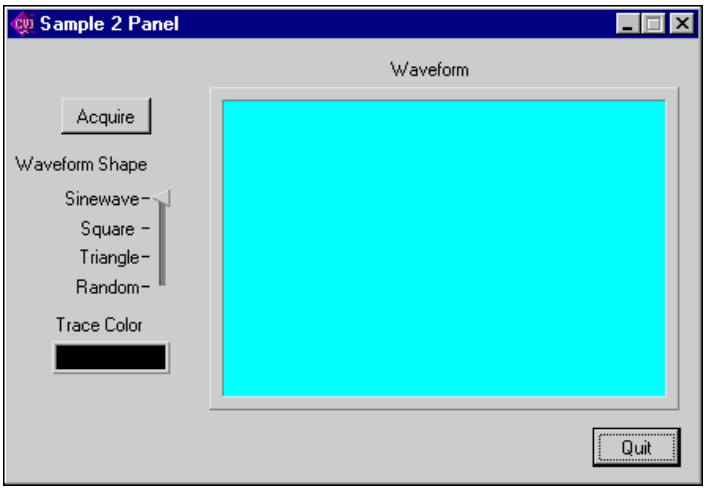

Figure 2-11. sample2.uir Panel When Running

3. Click **Acquire** to display a waveform on the graph control on the GUI. Move the slide control to select a new shape for the waveform. Click the color bar to choose a new color for the waveform trace. Finally, click **Acquire** after you change these settings. Repeat this step a few times, choosing different shapes and changing the color, then clicking **Acquire** to view your changes.
4. To halt program execution, click **Quit**.

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

You have completed the activities for this chapter. In Chapter 3, *Interactive Code Generation Tools*, you learn how to use interactive code generation tools.

3

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 get acquainted with some of the tools available for interactive code generation in LabWindows/CVI.

Setting Up

Follow these steps to prepare for this part of the tutorial.

1. Close all windows except the Project window.

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

2. Select **File»Open** and choose **Project (*.prj)** as the file type to open.
3. Find the `sample1` project file in the `tutorial` directory of your LabWindows/CVI installation and open it by double-clicking on `sample1.prj` in the dialog box.

Library Menu

All of the libraries in LabWindows/CVI are divided into a hierarchical structure under the **Library** menu as shown in Figure 3-1. This structure helps you access specific library functions easily.

Note Depending on which LabWindows/CVI options you have, the submenus under the **Library** menu include the **Analysis Library** or the **Advanced Analysis Library**.

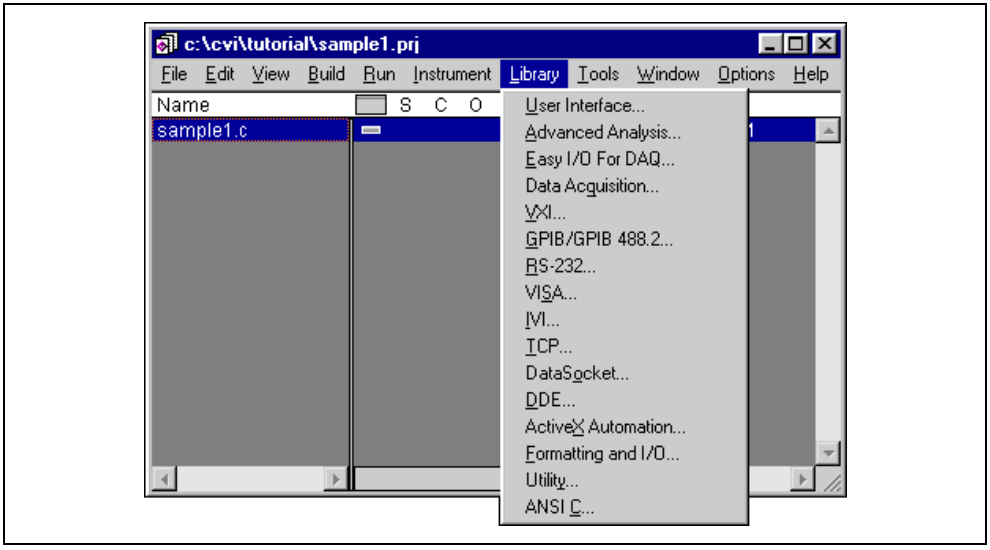

Figure 3-1. Library Menu

When you select one of the libraries from the **Library** menu, a function tree appears. With function trees, you can search quickly through the hierarchy of the library to find the right function. Figure 3-2 shows the User Interface Library function tree.

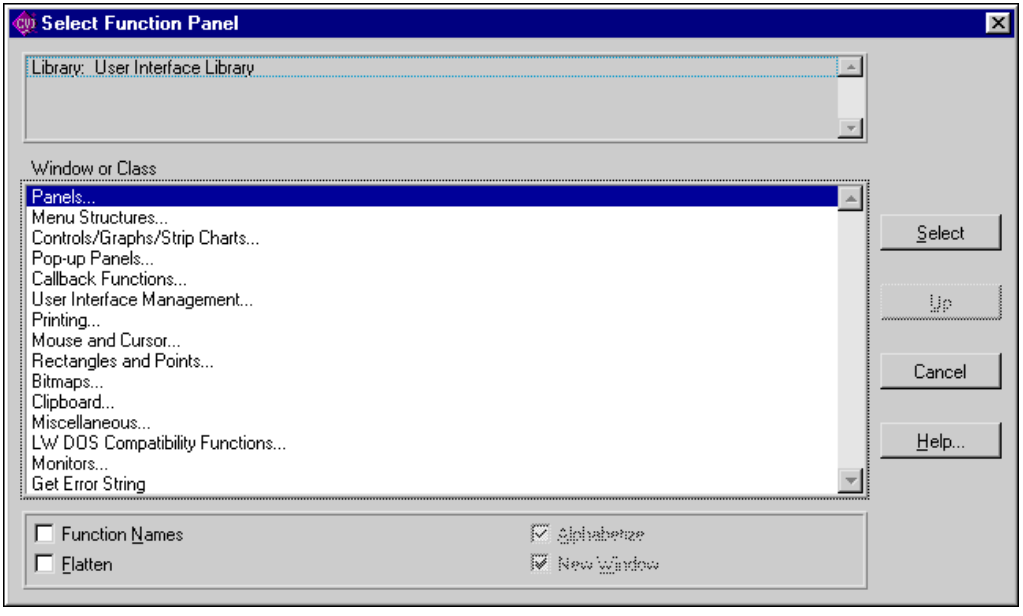

Figure 3-2. User Interface Library Function Tree

Accessing the User Interface Library

In this section of the tutorial, you use the User Interface Library to display a graph of the random numbers generated in `sample1`. Close the User Interface Library window if you opened it in the previous section and perform the following steps:

1. Double-click on the `sample1.c` filename in the Project window to view the source code.
2. Verify that the program runs correctly by selecting **Run»Debug sample1_dbg.exe**. Your 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`, position the cursor in the line above the `printf("Press the <Enter> key to terminate.....");` statement.
5. Select **Library»User Interface**.
6. In the dialog box that appears, select **Pop-up Panels** from the list, then press <Enter>.
7. In the next list that appears, select **Graph Popups**, then press <Enter>.
8. In the next list, select **Y Graph Popup**, then press <Enter>.

The window shown in Figure 3-3 appears.

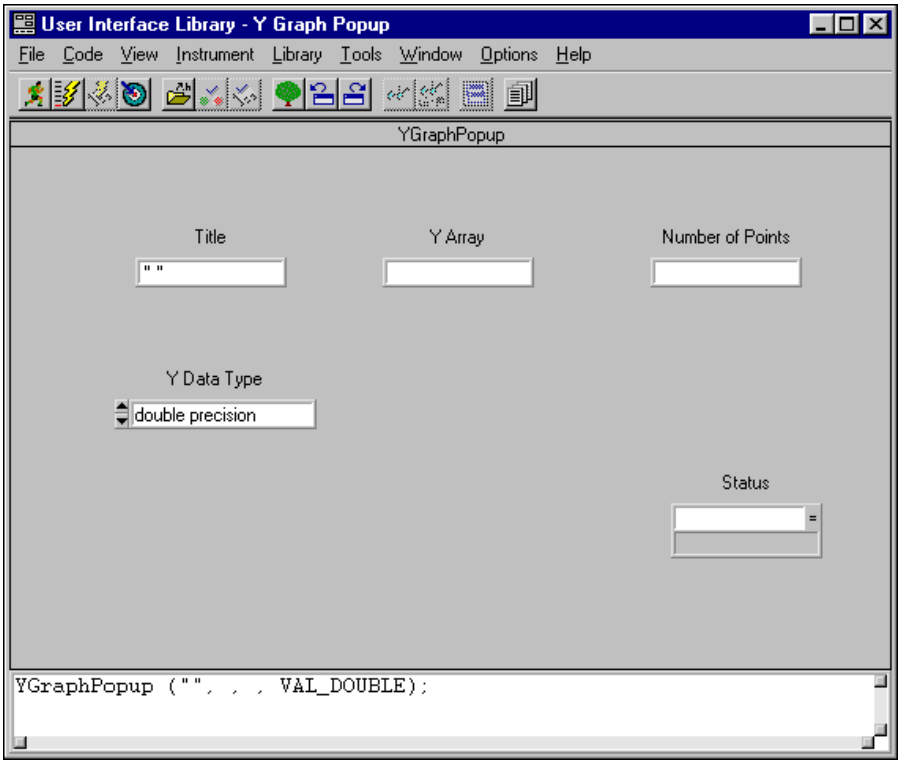

Figure 3-3. Y Graph Popup Function Panel

Function Panel Fundamentals

The display that appears 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 have online help to teach you the purpose of each function in the LabWindows/CVI libraries and the meaning of each parameter in the function call.
- With function panels, you automatically can declare variables in memory to be used as function parameters.
- With function panels, you can execute each LabWindows/CVI function interactively before incorporating it into your 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 items on the function panel are called controls, and you manipulate them to specify parameters. There are eight types of controls, and these controls are explained as you encounter them in the examples that follow.

Your 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.

Press the <Tab> key to move the cursor from one control to another. Alternatively, you can click a control to position the cursor.

Function Panel Help

Two types of help information are available on a function panel—general information about the panel and specific information about a control. Table 3-1 lists methods for accessing the help information.

Table 3-1. Panel Help Display Procedures

Type of Help	How to View Help
Function Help	Select **Help»Function**. *or* Right-click anywhere on the background of the Function panel.
Control Help	Place the cursor in the control, then select **Help»Control**. *or* Right-click on the desired control.

Select **Help»Function** to view information that pertains to the `YGraphPopup` function panel. Click **Done** to close the dialog box. Place the cursor in the **Y Array** control and press <F1> to view the control-specific help information. After reading the help information, close the dialog box.

Drawing a Graph

You now use the function panel to create a line of code that graphs the array of random numbers that the sample program generates. Remember, you can use the <Tab> key to move from one control to another, or you can click a control. Place your cursor in the **Title** control and perform the following steps.

1. Enter `"Random Data"` in the **Title** control. Your title must remain within the quotation marks.
2. Place your cursor in the **Y Array** control and enter the array name `datapoints`.

3. Place your cursor in the **Number of Points** control and enter the number `50`.
4. Place your cursor in the **Y Data Type** control. The up and down arrows show 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 on the ring control and select `double precision` from the pop-up menu.
5. Place your cursor in the **Status** control and enter the variable name `err`.

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

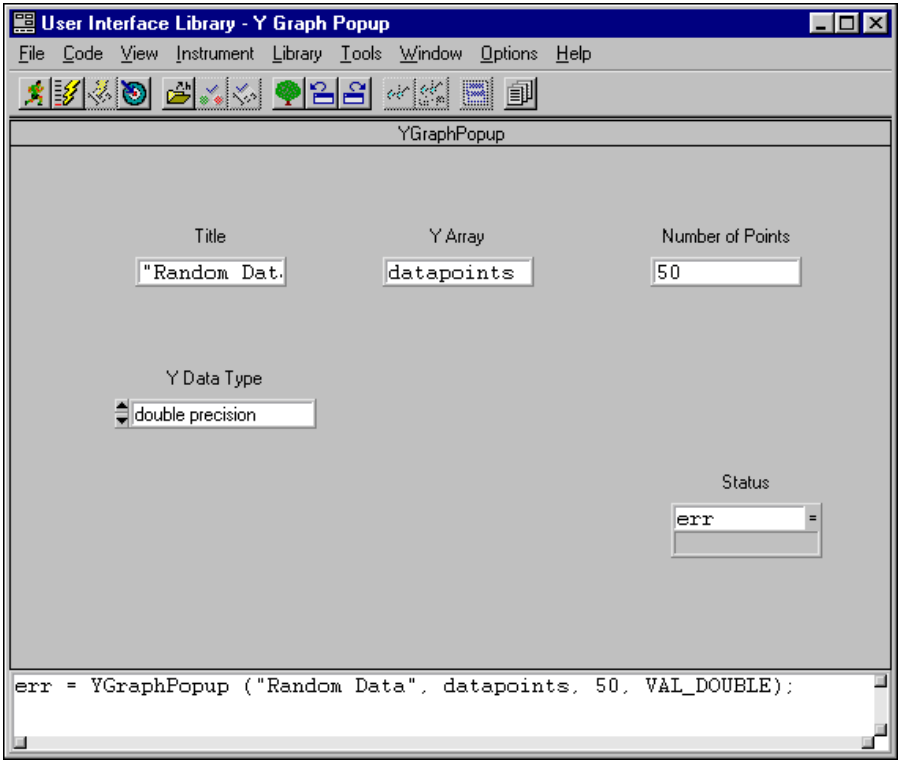

Figure 3-4. Completed Y Graph Popup Function Panel

Inserting Code from a Function Panel

Notice the small window at the bottom of the function panel. This is the Generated Code box. The line of code in the Generated Code box is generated while you manipulate the controls on the function panel. Place these lines of code directly into your source code by completing the following steps:

1. Select **Code»Set Target File**. In the dialog box that appears, select **sample1.c** from the list of windows.
2. Select **Code»Insert Function Call**. LabWindows/CVI pastes the code from the Generated Code box of the Function Panel window to the `sample1.c` source code at the position of the text cursor.
3. Close the `YGraphPopup` Function Panel window.
4. In the `sample1.c` source code, the `YGraphPopup` function appears where the cursor was located, as shown in Figure 3-5. Confirm that the new function appears immediately after the for loop.

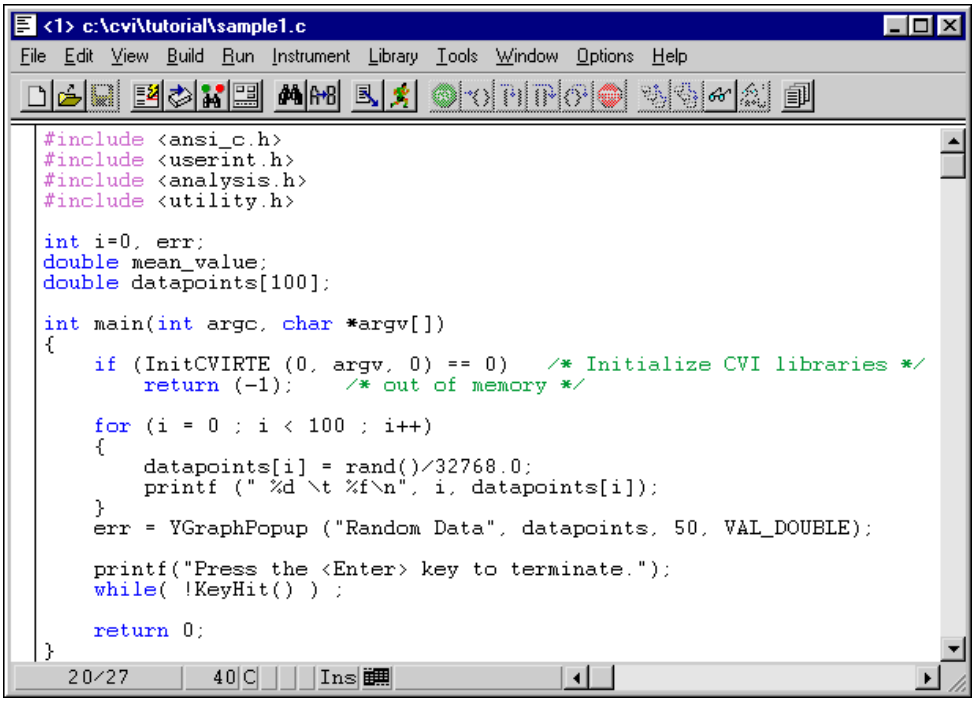

Figure 3-5. Source Window with Code Inserted from Function Panel

5. To execute the program, select **Run»Debug sample1_dbg.exe**. Click **Yes** if a dialog box prompts you to save changes before the project executes. As the program code executes, the Standard Input/Output window displays the screen output, then the graph of the data appears.
6. 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 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);
```

Perform the following steps to generate a call to the `Mean` function and insert the code into the appropriate area of the source code.

1. Select **View»Find Function Panel**.
2. In the dialog box that appears, type `Mean` into the **Find What** input box. Figure 3-6 shows the dialog box.

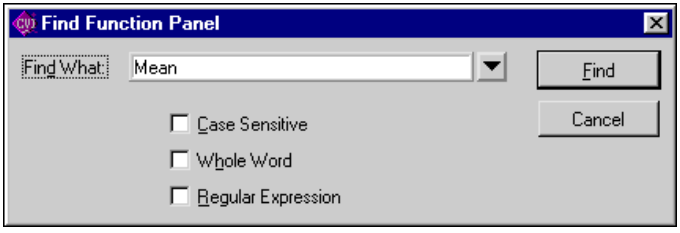

Figure 3-6. Find Function Panel Dialog Box

3. Click **Find** to go to the `Mean` function panel. The `Mean` function panel appears.
4. Confirm that your cursor is in the **Input Array** control and enter the array name `datapoints` in the **Input Array** control.
5. Move the cursor to the **Number of Elements** control and enter the number `100`.
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. Place your cursor in the **Mean** control and enter `&mean_value`. Confirm that your function panel window matches the one in Figure 3-7.

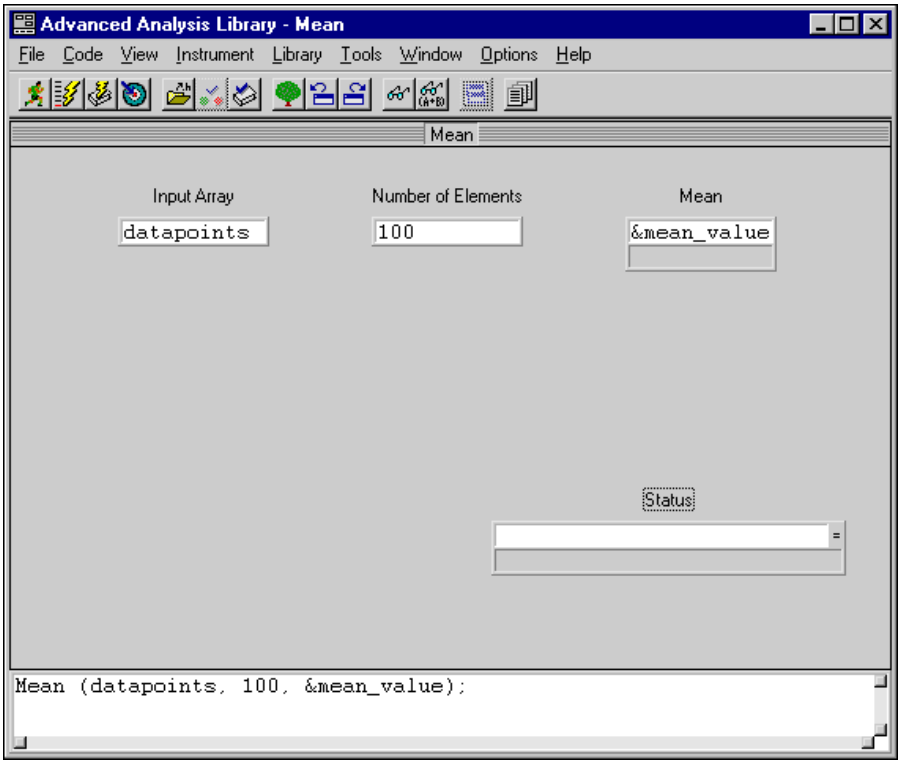

Figure 3-7. Completed Function Panel for Mean

2. In `sample1.c`, confirm your cursor is located above the `printf("Press the <Enter> key to terminate.");` statement.
3. In the function panel window for `Mean`, select **Code»Insert Function Call**, or press <Ctrl-i> to insert the line of code for calling the `Mean` function into `sample1.c`.
4. Close the `Mean` function panel. You see the code on the line where you left your cursor in the Source window.

Recalling a Function Panel

Notice that the function call to `YGraphPopup` graphs only 50 elements of the array `datapoints`. 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. Perform the following steps to edit this line of code using the Recall Function Panel feature.

1. Place the input 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 `YGraphPopup` function panel appears. Notice that the controls automatically reflect the state of the function call in the Source window.
3. Place your cursor in the **Number of Points** control and enter the number `100` in the control.
4. Insert the new code into your source code by selecting **Code»Insert Function Call**. The Replace/Insert dialog box appears as shown in Figure 3-8.

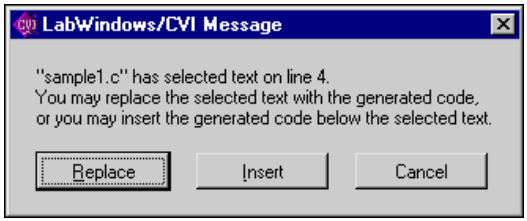

Figure 3-8. Replace/Insert Dialog Box

5. Click **Replace**, close the Function Panel window, and return to the Source window.

Notice that in `sample1.c` the call to `YGraphPopup` is now set to graph 100 elements of the array `datapoints`.

Finishing the Program

Now you have a program that generates 100 random numbers, plots the numbers on a graph, and calculates the mean value. As a final step, type the following line to the end of the main function in the Source window.

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

Confirm that your completed source coded 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 (InitCVIRTE (0, argv, 0) == 0)   /* Initialize CVI libraries */
        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;
}
```

Execute the program by selecting **Run»Run Project**. If a dialog box prompts you to save changes before running your program, click **Yes** or press <Enter>. 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. Finally, it calculates the mean of the numbers and prints it in the Standard Input/Output window after the last output line.

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 your program. You also learned that function panels can teach you how functions operate through the online 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 the Libraries*, and Chapter 7, *Adding Analysis to Your Program*, you learn how to use function panels to declare variables for use in your program and how to run functions interactively.

This concludes this chapter of the tutorial. In the next chapter you learn how to use the executing and debugging tools available in LabWindows/CVI.

4

Executing and Debugging Tools

In this chapter, you 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

Setting Up

1. Open the `sample3.prj` project in the `tutorial` directory.
2. Close all windows except the Project window for `sample3.prj`.

   If you did not save the previous contents of these windows, LabWindows/CVI prompts you to do so. If you want to save the contents of a previous window, click **Save** and enter a file name into the **File Name** control. If you do not want to save the previous window contents, click **Discard**.
3. Double-click the `sample3.c` filename in the project list to open the Source window.

Figure 4-1 shows the program `sample3.c` as it appears in the Source window.

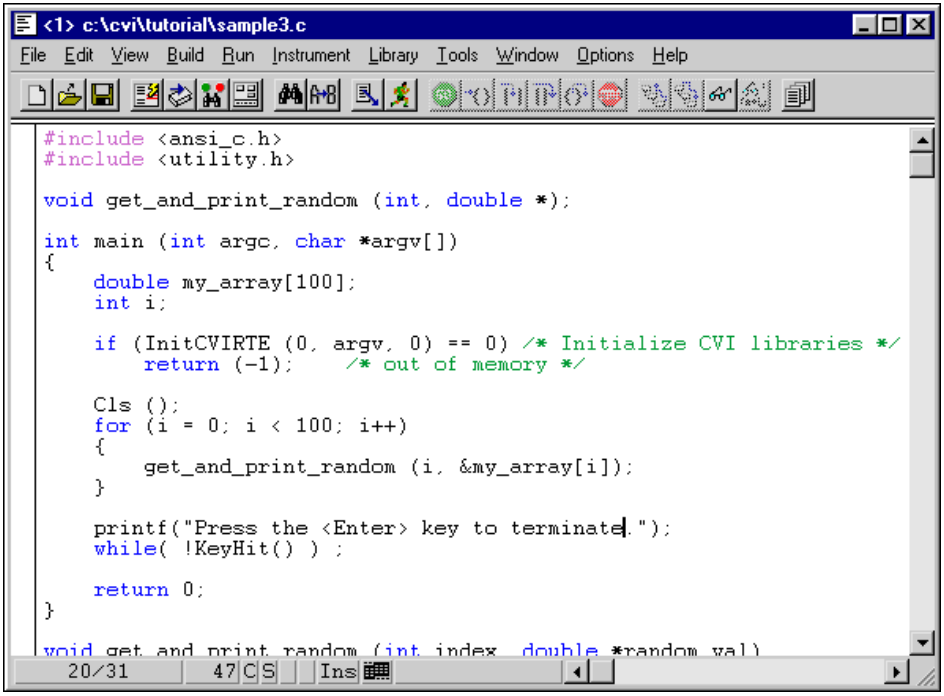

Figure 4-1. sample3.c Program

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

The difference between the programs is that in `sample3.c`, you place the steps of assigning a random number and printing out the values in a function. This arrangement of program instructions serves to illustrate the debugging tools available in the LabWindows/CVI interactive development environment.

Step Mode Execution

Step mode execution is a useful run-time tool for debugging programs. In step mode, you can execute a program in steps described as follows.

1. Select **Run»Break at First Statement** to stop execution at the first line in the source code. When this mode is activated, a checkmark appears next to the **Break at First Statement** command in the **Run** menu.

2. **Run»Debug sample3_dbg.exe** to begin execution of the program. After the program compiles, the `main` function line in the program appears highlighted in the Source window, indicating that program execution is currently suspended.
3. To execute the highlighted line, select **Run»Step Into**.

   To avoid accessing the **Run** menu each time you perform step mode execution, use the shortcut key combinations listed in Table 4-1. Remember that you 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.
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.
Terminate Execution	<Ctrl-F12>		Halts execution of the program during step mode.

4. To find the definition of the `get_and_print_data` function, double-click the call to the function on line 17 of `sample3.c`, then select **Edit»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 be listed in the Project window. You can use this command also to find variable declarations.

5. Use **Step Into** to begin stepping through the program. Notice that when the function call `get_and_print_random` 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.

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 four ways:

- **Programmatic Breakpoint**—You insert a Breakpoint icon in the Source window.
- **Manual Breakpoint**—You press <Ctrl-F12> while a window is active in the LabWindows/CVI environment.
- **Breakpoint on Error**—You cause LabWindows/CVI to pause when a library function returns an error.
- **Conditional Breakpoint**—You cause LabWindows/CVI to pause when a user-specified condition becomes true.

The following sections explain programmatic breakpoints and manual breakpoints. For additional information on conditional breakpoints and the Breakpoint on Error feature, refer to the *Run Menu* section of Chapter 5 *Source and Interactive Execution Windows*, in the *LabWindows/CVI User Manual*.

Programmatic Breakpoints

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

1. Halt program execution if you have not already done so, by selecting **Run»Terminate Execution**.
2. Disable **Run»Break at First Statement**. The checkmark next to the **Break at First Statement** menu item disappears.
3. Click in the left-hand column of the Source window to the left of the line that contains the following statement:

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

A red diamond, that represents a breakpoint appears beside that line as shown in Figure 4-2.

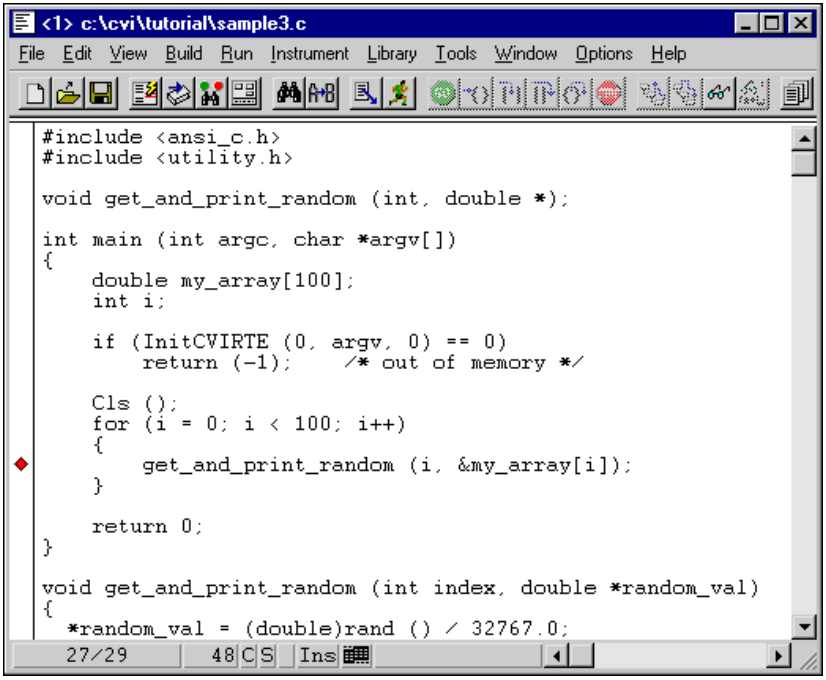

Figure 4-2. Breakpoint beside a Line of Code

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. Halt the program at a breakpoint by pressing <Ctrl-F12>. Alternatively, you can halt program execution by selecting **Run»Terminate Execution**.
7. Remove the breakpoint from the program by clicking the red diamond so that it disappears.

Manual Breakpoints

The following steps describe how you can enter a breakpoint after program execution has begun using the quick keys <Ctrl-F12>.

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 when the breakpoint symbol was encountered. 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 **Program»Terminate Execution**, to halt 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 has the following special displays for viewing and editing the data for your program.

- Variables Display window
- Array Display window
- String Display window
- Watch window, a special utility that lets you view variable and expression values during program execution.

Variables Window

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

When you select **Variables**, the display appears as shown in Figure 4-3.

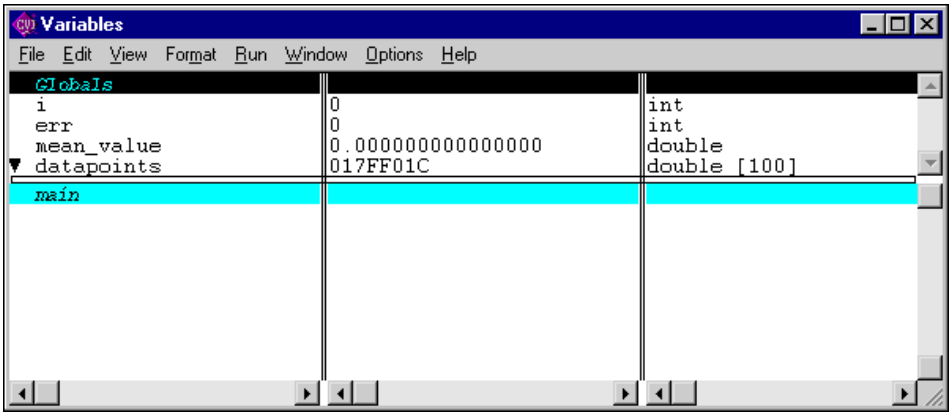

Figure 4-3. Variables Window

The Variables window lists the name, value, and type of each variable currently declared. Variables are displayed in categories according to how they are defined, such as global or local, and in which file they are defined. The display in Figure 4-3 shows a number of global variables that are currently declared.

You can view the Variables window at any time to inspect variable values. This feature is especially useful when you are stepping through a program during execution that has stopped at a breakpoint. Perform 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 at First Statement**. A checkmark appears beside the **Break at First Statement** command in the **Run** menu.
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**. The Variables window appears as shown in Figure 4-4.

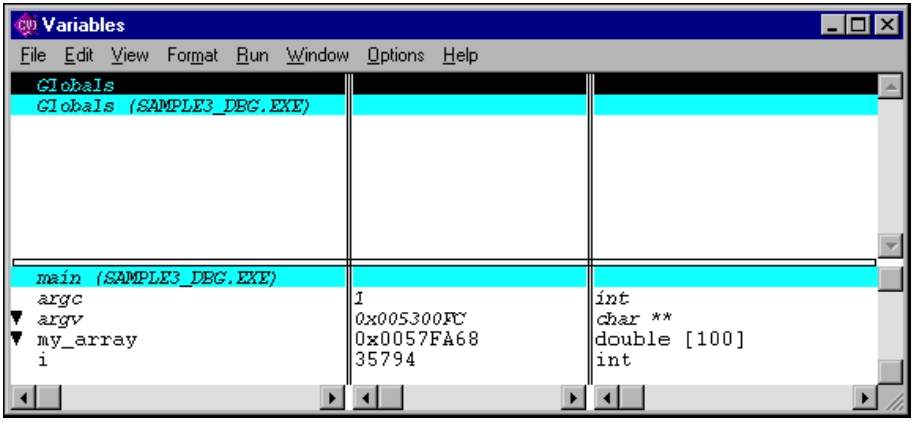

Figure 4-4. Variables Window during Execution of main

Notice that variables now appear under a heading called *main*: a double-precision array (`my_array`) and an integer (`i`). The section called *main* displays all variables that are declared locally to the `main` function in the Source window.

Note The values you see for your project might differ from the values shown in the preceding illustration.

4. Close the window and return to the Source window.
5. Select **Run»Step Into** until the highlighting comes to the line `*random_val = (double)rand () / 32767.0;`, which is the first statement in the function `get_and_print_random`.

6. Select **Window»Variables**. The Variables window appears as shown in Figure 4-5.

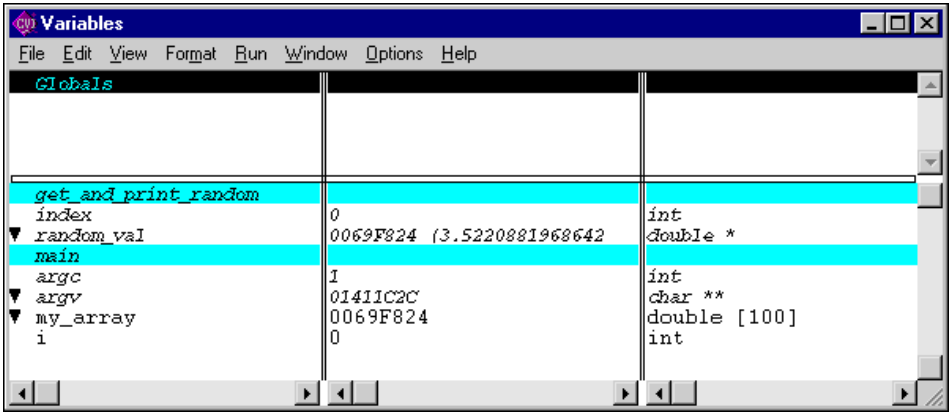

Figure 4-5. Variables Window during Execution of get_and_print_random

Notice that a new section appears under the heading of the function `get_and_print_random` to show the variables that are declared locally to the function.

7. Click the Source window in the background to make it the active window.

The Variables window groups variables according to their scope (global or local) and the program module in which they are declared (`main` function or a subroutine).

Leave the program in breakpoint mode and continue with the next example.

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.

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

```
*random_val = (double)rand () / 32767.0;
```

Perform the following steps.

1. Select **Run»Step Into** until the `for` loop executes a few times and the highlight appears on the following statement.

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

2. Highlight the `i` variable and select **Run»View Variable Value** to display the Variables window with the `i` variable highlighted as shown in Figure 4-6.

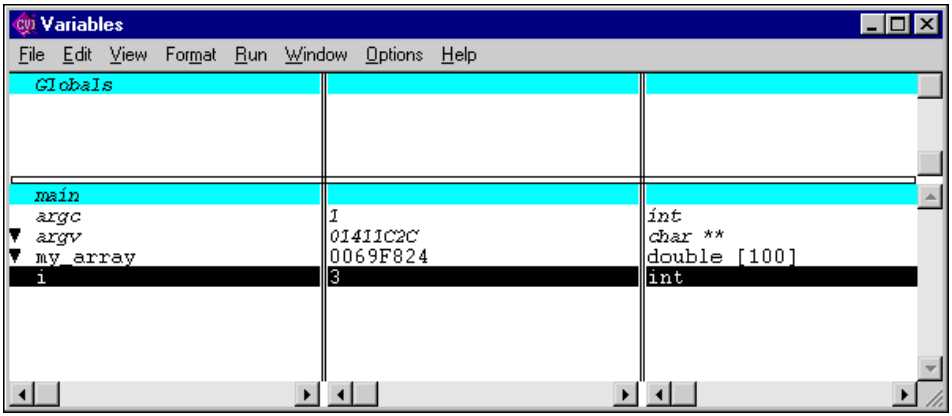

Figure 4-6. Variables Window with i Variable Highlighted

3. Press <Enter> to display the Edit Value dialog box, shown in Figure 4-7, and enter the value `10`.

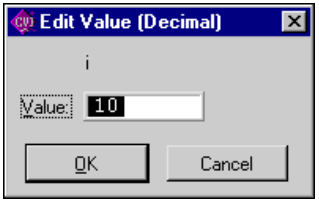

Figure 4-7. Edit Value Dialog Box

4. Press <Enter>. Notice that the value of `i` is now 10 in the Variables window.
5. Click the Source window in the background to make it the active window.
6. Select **Run»Step Into**.
7. Select **Window»Standard Input/Output** or click the Standard Input/Output window to make it the active window.

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

Leave the program in breakpoint mode and continue with the next example.

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 manner as you edited variables using the Variables window.

Confirm the program is still be in breakpoint mode. Perform the following steps.

1. Click the Source window to make it active.
2. Select **Run»Step Into** or press <F8> until the highlight reaches the following line.

   ```
   printf (" %d \t %f \n", *index, *random_val);
   ```

3. Click the Variables window in the background to make it the active window.
4. Position the highlight on the array `my_array` under the heading main and press <Enter> or double-click the variable name `my_array`. The display shown in Figure 4-8 appears on your screen.

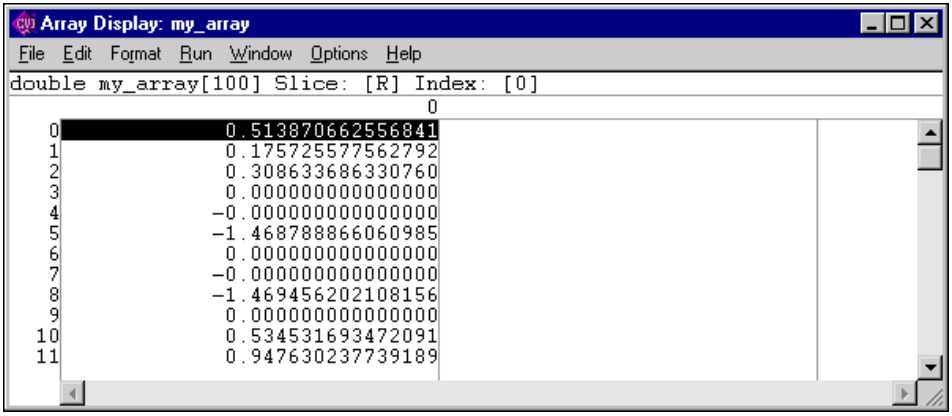

Figure 4-8. Array Display

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

The Array Display shows the values of array elements in tabular format. In Figure 4-8, 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 (`0`).

You can scroll through the Array Display by using the arrow keys or the scroll bar on the right edge of the display. 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 the 12th element of the array, follow these steps.

1. Either double-click the 12th element or highlight the 12th element (index 11) in the array and press <Enter>. The dialog box shown in Figure 4-9 appears.

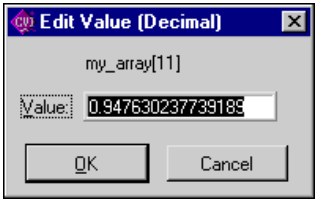

Figure 4-9. Edit Value Dialog Box

2. Enter the value `0.5` and press <Enter>. Notice that the 12 element of the array is now set to 0.5.
3. Close the Array Display and return to the Variables window.
4. Click the Source window in the background to make it the active window.
5. 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.
6. Press <F5>, or select **Run»Continue**, to complete program execution.

String Display

Another useful data display is the String Display. When you select a string variable from the Variables window, the String Display appears. 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 just performed in the Array Display. Chapter 8, *Array and String Display Windows*, in the *LabWindows/CVI User Manual*, describes the String Display in more detail. An example String Display appears in Figure 4-10.

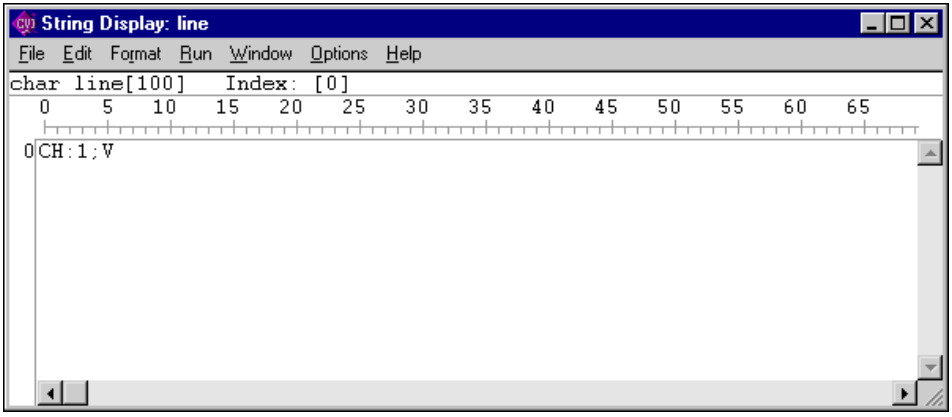

Figure 4-10. String Display

Watch Window

The Watch window is a powerful debugging tool because you can view values of variables changing dynamically as your program executes. You can also use the Watch window to view expression values, and set conditional breakpoints whenever 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, select **Run»Break at First Statement** so that a checkmark appears next to it.
2. Select **Run»Debug sample3_dbg.exe**, or press <Shift-F5>, to start program execution. Execution breaks with the `main` function highlighted.
3. Press <F8> until the highlight reaches the following line:

   ```
   void get_and_print_random (int index, double *random_val);
   ```

4. Select **Window»Variables** to open the Variables window, or click the Variables window if it is already active in the background.
5. Click the `random_val` variable or press the <Down> arrow key until the `random_val` variable is highlighted.

6. 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 4-11 appears.

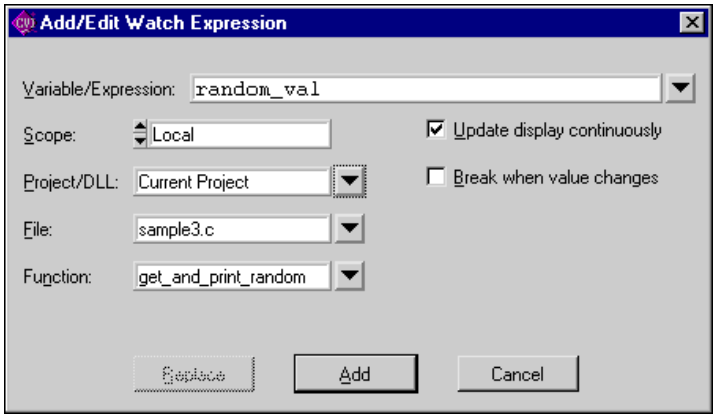

Figure 4-11. Add/Edit Watch Expression Dialog Box

7. Click the checkbox next to **Update display continuously** if it is not already checked and then click **Add**.
8. Click the Source window in the background to make it the active window. Position and size the Watch window so that you can see it in addition to the Source window.
9. 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.
10. When program execution ends, close all windows except the Project window.

This chapter acquainted you with the LabWindows/CVI programming environment. In the next part of the tutorial, Chapter 5, *Building a Graphical User Interface*, Chapter 6, *Using Function Panels and the Libraries*, Chapter 7, *Adding Analysis to Your Program*, and Chapter 8, *Using an Instrument Driver*, you build an actual project in LabWindows/CVI.

Part II

Building an Application in LabWindows/CVI

- Chapter 5, *Building a Graphical User Interface*, contains instructions for building a project consisting of a Graphical User Interface and a C source file.
- Chapter 6, *Using Function Panels and the Libraries*, contains instructions for using LabWindows/CVI function panels to generate code. You will then use this code to plot the graph control array on the user interface that you built in Chapter 5.
- Chapter 7, *Adding Analysis to Your Program*, contains instructions for adding a simple analysis capability to your program to compute 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.
- Chapter 8, *Using an Instrument Driver*, contains instructions for using a simple instrument driver from the LabWindows/CVI Instrument Library. 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 session 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.

5

Building a Graphical User Interface

This chapter and Chapter 6, *Using Function Panels and the Libraries*, Chapter 7, *Adding Analysis to Your Program*, and Chapter 8, *Using an Instrument Driver*, describe how to build a project that consists of a GUI and a C source file. Chapter 9, *Additional Exercises*, contains additional exercises to practice and expand on what you have learned.

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 develop a simple project that consists of a GUI controlled by a C source file. In this chapter, you learn to design a user interface with the User Interface Editor. In subsequent chapters, you create a simple C source file that operates the user interface.

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 in a panel. In the Chapter 6, *Using Function Panels and the Libraries*, you learn about the User Interface Library, which includes a set of functions to control the interface programmatically.

Setting Up

Perform the following steps to set up this part of the tutorial.

1. Select **File»New»Project (*.prj)**. A dialog box prompts you to unload the current project. Press <Enter> or click **Yes** to unload the project.
2. The dialog box shown in Figure 5-1 appears. Click **OK** to maintain the current configuration of the LabWindows/CVI environment for the new project.

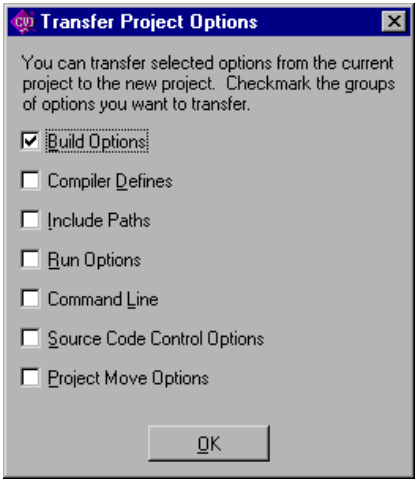

Figure 5-1. Transfer Project Options Dialog Box

3. Close all LabWindows/CVI windows except the Project window.

User Interface Editor

The User Interface Editor is an interactive drag-and-drop editor that you use to design 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 your user interface in the User Interface Editor window, you can begin to write your C source code to control the GUI. You assign a name to each panel, menu, and control on your user interface. Use that name in the C source code to differentiate the controls on the GUI. You also can assign a function name to controls on your user interface that is called automatically whenever you operate that control during program execution. You associate a constant name and a callback function with a particular 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 resource file (`.uir`), LabWindows/CVI automatically generates an include file that defines all the constants and callback functions you have assigned.

CodeBuilder

After you complete your `.uir` file, the CodeBuilder utility in LabWindows/CVI can help you create a complete source file. CodeBuilder automatically creates a source file based on the callback functions specified in your `.uir` file.

Sample Project

In the next few chapters of this tutorial, you follow instructions to 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 the Libraries*).
4. Develop your own callback function to compute the mean value of the waveform (Chapter 7, *Adding Analysis to Your Program*).
5. Incorporate the use of an instrument driver in the project to simulate data acquisition (Chapter 8, *Using an Instrument Driver*).

Building a User Interface Resource (.uir) File

The following steps teach you about building a user interface. You insert an additional command button and add a graph control to the user interface of the sample project.

Opening a .uir File

1. Select **File»Open» User Interface (*.uir)**. Select `template.uir` from the file list. The User Interface Editor appears with the controls as shown in Figure 5-2.

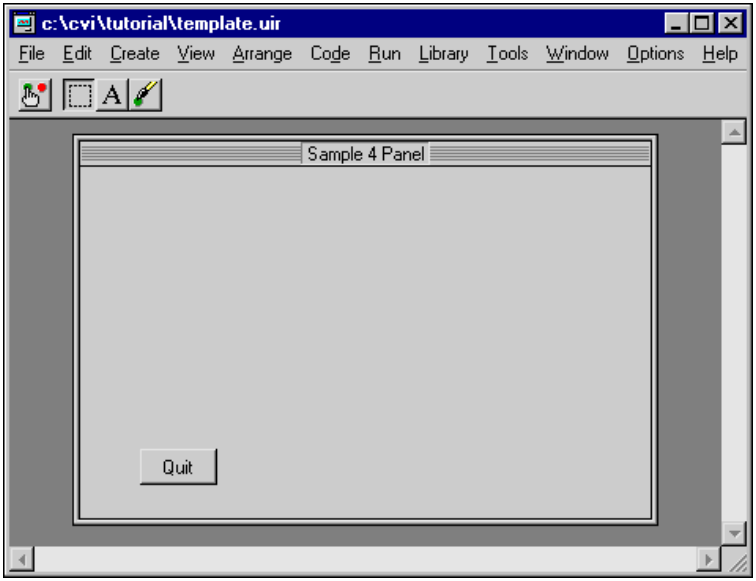

Figure 5-2. User Interface Editor with GUI Panel

Use the commands in the **Edit** menu to cut, copy, paste, align, and space user interface controls in the editor.

Use the commands in the **Create** menu to place user interface objects—such as numeric controls, LEDs, command buttons, toggle switches, graphs, and strip charts—onto your user interface file.

2. Notice that the user interface is already partially completed. You must add a button to trigger an acquisition and a graph control to display the acquired waveform so that it looks like the `.uir` in Figure 5-3. You add the button and graph control in the steps that follow.

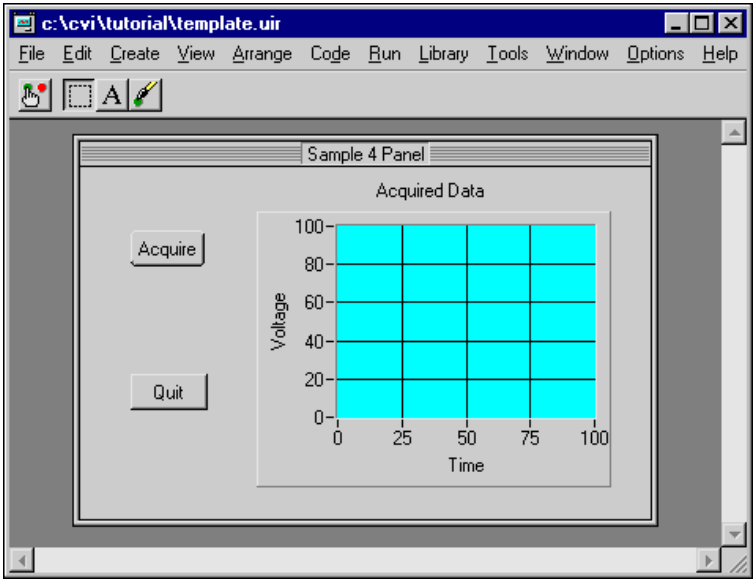

Figure 5-3. User Interface Editor with Completed GUI Panel

Adding a Command Button

1. Select **Create»Command Button** and choose one of the button styles shown in the submenu. A button labeled **OK** appears on the panel.
2. Drag-and-drop the new button near the Quit button on the panel. If you do not have a mouse, press the <Tab> key until the **OK** button is selected and then use the arrow keys to position the button.

3. To edit the button attributes, double-click the button or press <Enter>. The Edit Command Button dialog box appears as shown in Figure 5-4.

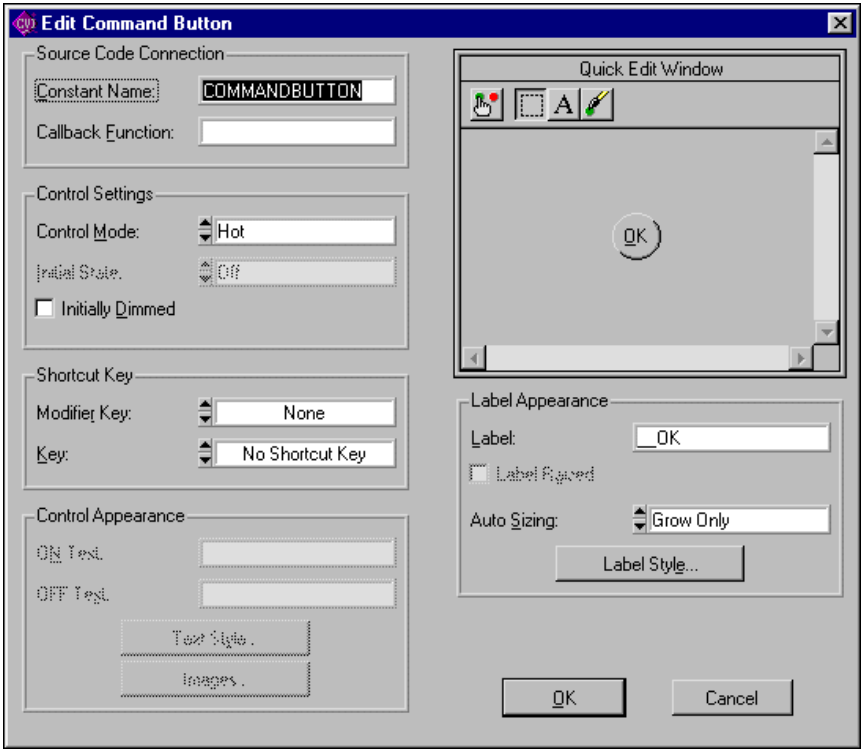

Figure 5-4. Edit Command Button Dialog Box

4. Assign a constant name to the button. Your C source code uses this constant name to communicate with the button. LabWindows/CVI creates a default name for you. You can assign your own constant names to your `.uir` file. Type `ACQUIRE` in the **Constant Name** control within the Source Code Connection section of the dialog box. Use all capital letters.
5. Assign a function name for your program to call when a user clicks the Acquire button. Type `AcquireData` in the Callback Function control. In the next chapter you write the source code for the `AcquireData` function. Make sure the Source Code Connection section of the dialog box looks exactly as shown in Figure 5-5 to ensure that your program can properly communicate with the button.

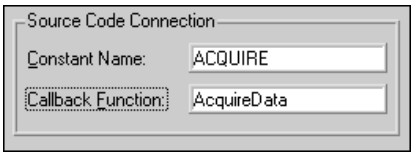

Figure 5-5. Source Code Connection Section of Edit Command Button Dialog Box

6. Place your cursor in the **Label** control within the Label Appearance section of the dialog box. To change the label on the command button, type `Acquire` in place of the existing characters, `__OK`, in the **Label** control.
7. (Optional) Click **Label Style** to open another dialog box and make custom settings for your button. Click **OK** when you finish.
8. Click **OK** at the bottom of the Edit Command Button dialog box or press <Enter>. Confirm that your user interface looks similar to the one shown in Figure 5-6.

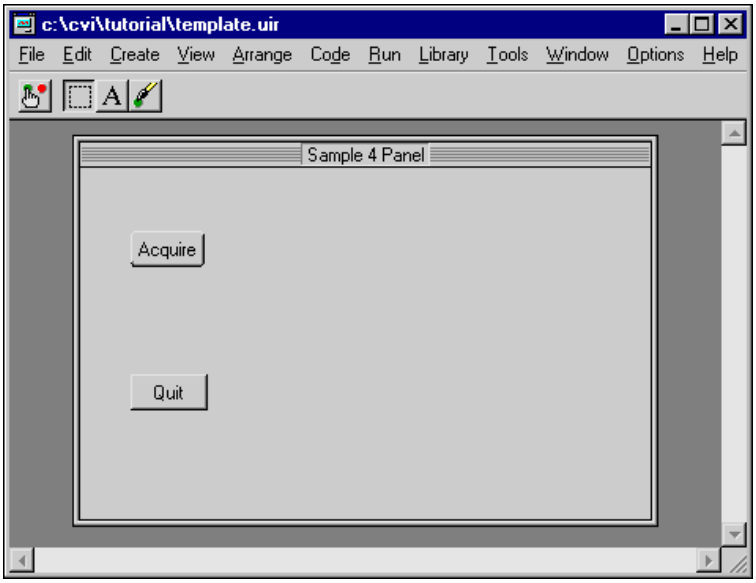

Figure 5-6. User Interface with Two Buttons

Adding a Graph Control to the User Interface

1. Select **Create»Graph»Graph**. A graph control named Untitled Control appears on your user interface.
2. Drag-and-drop the graph with the mouse. To size the graph, click and drag one of the corners.

3. Double-click the graph control to display the Edit Graph dialog box in which you customize the graph attributes.
4. Type `WAVEFORM` in the **Constant Name** control within the Source Code Connection section of the dialog box. Use all capital letters.

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 within the Label Appearance section of the dialog box. Confirm that the dialog box looks exactly as shown in Figure 5-7.

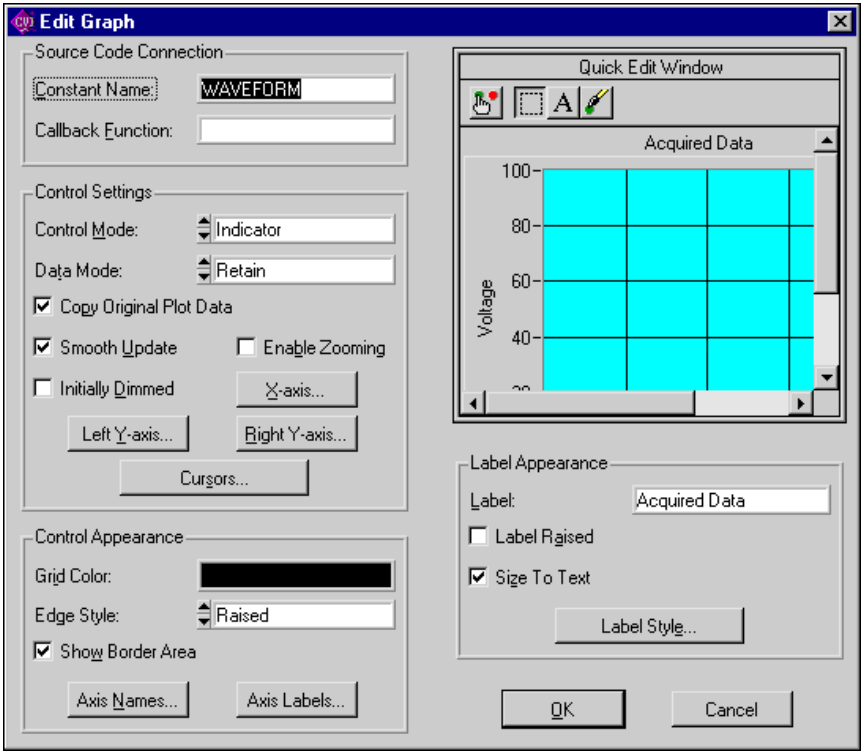

Figure 5-7. Edit Graph Dialog Box

6. (Optional) Use the **X-axis** and **Left Y-axis** buttons to display the Edit Axis Settings dialog box. Assign `Time` and `Voltage` labels to the x- and y-axis labels respectively and click **OK**.
7. After you set the graph attributes, click **OK** at the bottom of the Edit Graph dialog box to close the dialog box.

Your completed user interface looks like the one shown in Figure 5-3.

Saving the .uir File

1. Select **File»Save As** to save the `.uir` file with your new controls. Name your file `sample4` and click **Save** to save the file and close the File Save dialog box.
2. Select **View»Preview User Interface Header File** to inspect the header file that LabWindows/CVI has automatically created. It appears as shown in Figure 5-8.

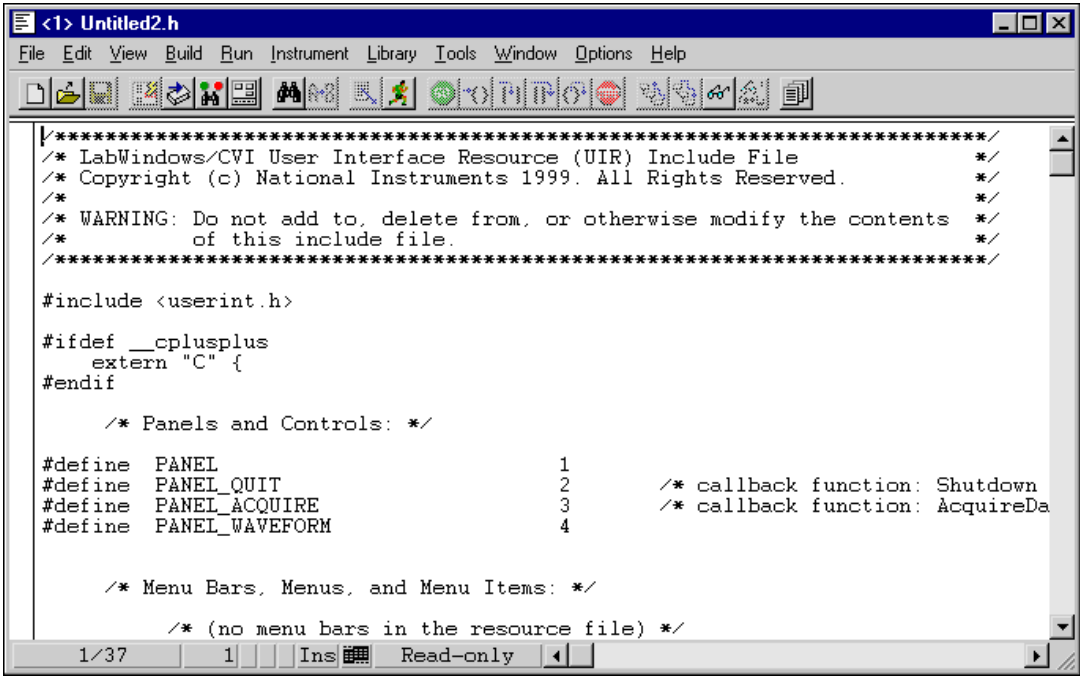

Figure 5-8. User Interface Header File

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**. The dialog box shown in Figure 5-9 appears, with a checkmark beside the `EVENT_COMMIT` callback event.

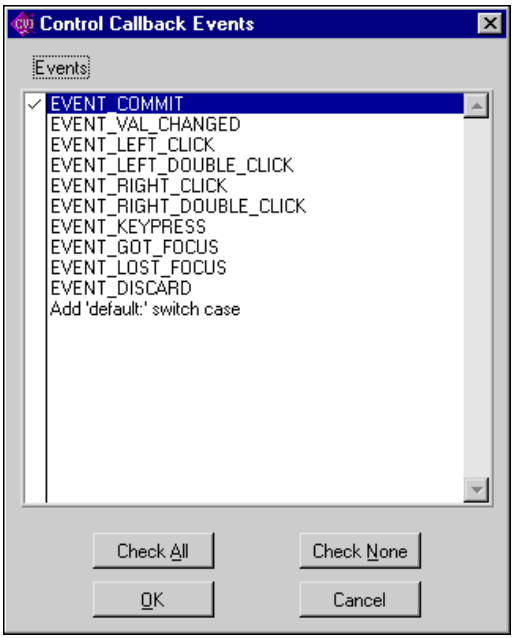

Figure 5-9. Control Callback Events Dialog Box

2. Later in this tutorial, you develop code to display help information 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. You access the functionality of CodeBuilder in the **Code** menu of the User Interface Editor window. Select **Code»Generate»All Code** to display the Generate All Code dialog box of CodeBuilder shown in Figure 5-10. You need to specify several options in this dialog box.

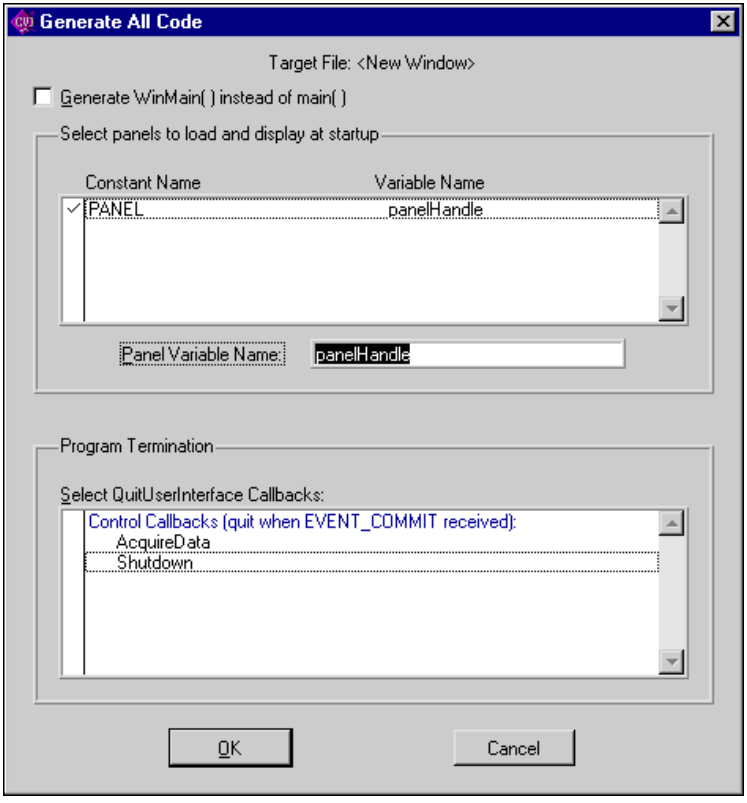

Figure 5-10. Generate All Code Dialog Box

4. First, you must decide which panels to display at program startup. In this case, only one user interface panel exists. Select this item and use the mouse or press <Space bar> to place a checkmark beside the filename so that this file displays at program startup.

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

5. The lower half of the Generate All Code dialog box shows a list of callback functions in your `.uir` file. You can select a function from this list that causes the program to terminate execution. In this case, select the **Shutdown** function in the dialog box so that a checkmark appears next to it.

6. Click **OK**. This action causes CodeBuilder to build the source code for your program. A new Source window appears 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 Shutdown (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 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;
}
```

7. Select **File»Save** in the Source Window and name the file `sample4.c`.

You have completed the activities for this chapter. In the Chapter 6, *Using Function Panels and the Libraries*, you complete the C source code for displaying a waveform on the graph.

6

Using Function Panels and the Libraries

In this chapter of the tutorial, you use LabWindows/CVI function panels to generate code. You then use this code to plot the graph control array on the user interface that you built in Chapter 5, *Building a Graphical User Interface*. If you have not completed Chapter 5, go back and do so now.

Setting Up

If you did not directly proceed from Chapter 5, *Building a Graphical User Interface*, follow these steps to set up LabWindows/CVI so that you can complete this part of the tutorial.

1. Close all windows other than the Project window by selecting **File»Close**.
2. Select **File»Open»Source (*.c)**.
3. Type `sample4.c` in the **FileName** control, or select it from the dialog box.
4. If **Break at First Statement** is checked on the **Run** menu, turn off this option by selecting **Run»Break at First Statement**.
5. Select **File»Open»User Interface (*.uir)**.
6. Type `sample4.uir` in the **FileName** control or click the name in the dialog box.
7. Minimize the `sample4.uir` window for later use.

Analyzing the Source Code

The source code for the `sample4` program is incomplete. In this chapter, you add a line of 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 very 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 panels 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.

AcquireData Function

The `AcquireData` function automatically executes whenever you click **Acquire** on the user interface. At this time, the `AcquireData` function generates an array of random data. 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;
}
```

Shutdown Function

The Shutdown 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 halts execution of the program. The Shutdown function is shown in the following code.

```
int CVICALLBACK Shutdown (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

Your task is to complete the source code for sample4.c so that 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 array within a for loop.

1. Declare the array variable datapoints and index variable i at the top of the source window by entering the following lines of code.

   ```
   int i;
   double datapoints[100];
   ```

2. Position the input cursor in the Source window on the blank line following the case EVENT_COMMIT in the AcquireData function.

3. LabWindows/CVI utilities help you generate code for common C constructs such as `for loops`, `while loops`, and `switch statements`. Select **Edit»Insert Construct»For Loop** to display the dialog box shown in Figure 6-1.

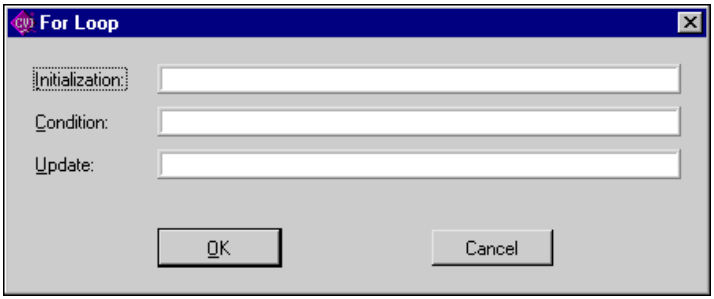

Figure 6-1. For Loop Dialog Box

4. Enter the following values in the For Loop dialog.

   Initialization: i=0

   Condition: i<100

   Update: i++

   Click **OK**.

5. Enter the following line of code within the for loop construct to generate the random numbers.

```
datapoints[i] = rand()/32767.0;
```

Finding the PlotY Function

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

1. Position the input cursor in the Source window on the blank line following the closing bracket just after the `datapoints[i] = rand()/32767.0` function call within the `AcquireData` function.
2. Select **Library»User Interface** to display the dialog box shown in Figure 6-2.

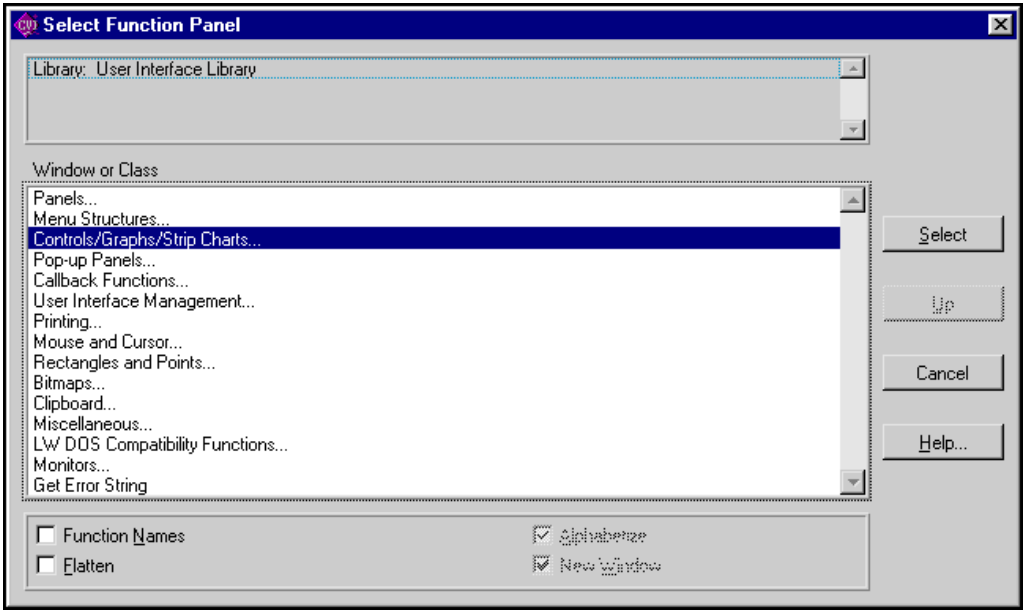

Figure 6-2. Select Function Panel Dialog Box

3. Select **Controls/Graphs/Strip Charts** and press <Enter>.
4. Select **Graph and Strip Charts** from the list and press <Enter>.

5. Press <Enter> to select **Graph Plotting and Deleting** from the list to display all the LabWindows/CVI functions related to displaying or operating data on graphs and strip charts, as shown in Figure 6-3.

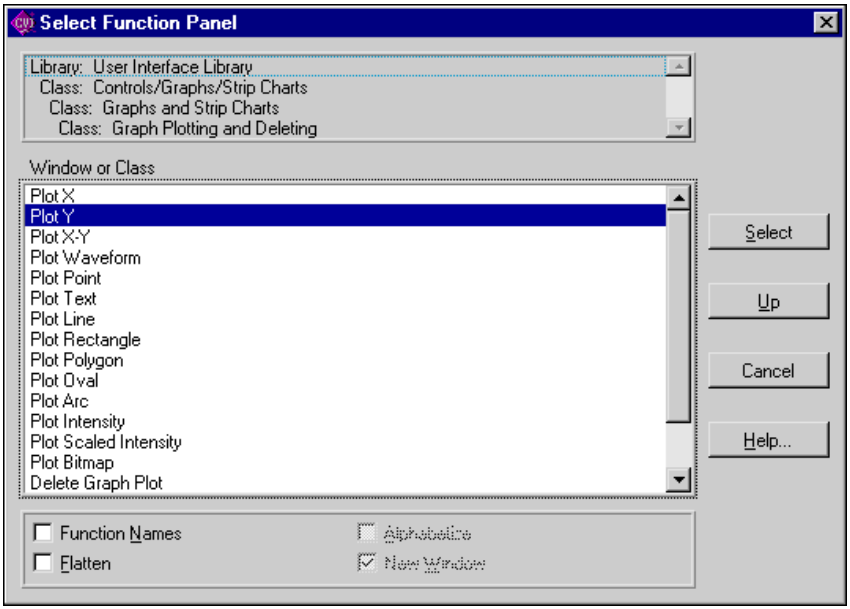

Figure 6-3. Graph Plotting and Deleting Functions

6. Use the <Down> arrow key or click to select the `PlotY` function and press <Enter>. The function panel for `PlotY` appears as shown in Figure 6-4.

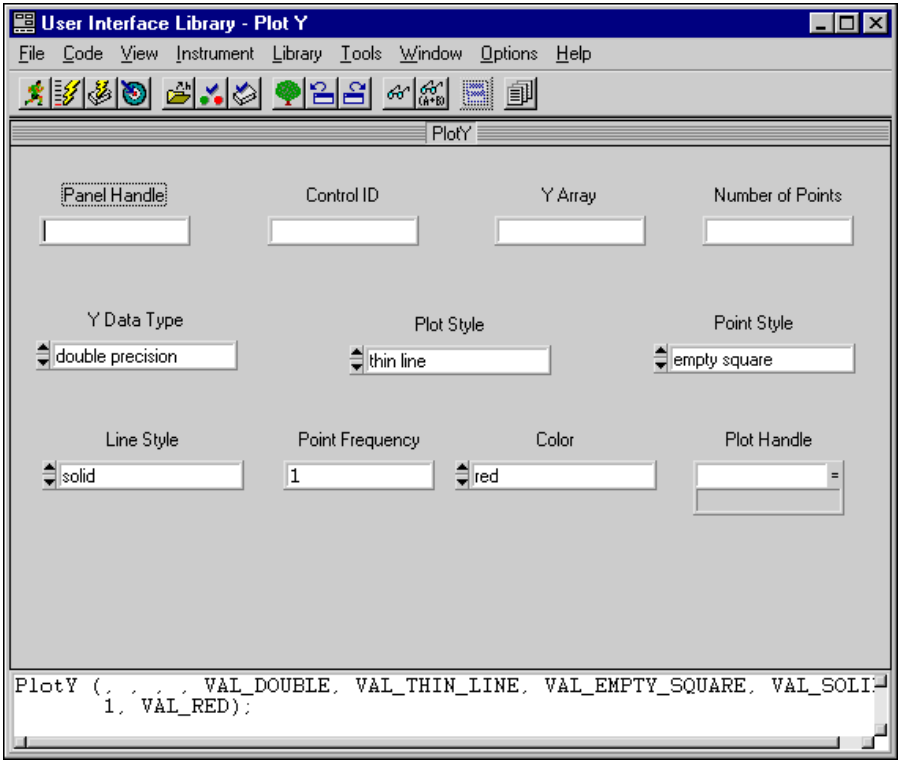

Figure 6-4. Plot Y Function Panel

Building the PlotY Function Call Syntax

In this section, you use the `PlotY` function panel to learn about the operation of the `PlotY` function, automatically generate the source code for the function call, and insert the function call into your program.

Follow these steps to build the function call using the `PlotY` function panel.

1. Place your cursor in the **Panel Handle** control. Choose **Code»Select Variable**. A list of variable names used in your program appears. Click on `panelHandle` in the list.
2. The **Control ID** control contains the constant name assigned to the graph control. To get a complete list of all the constant names in the `.uir` file you are working on, select **Code»Select UIR Constant**. A list of all constant names in the `.uir` file appears as shown in Figure 6-5.

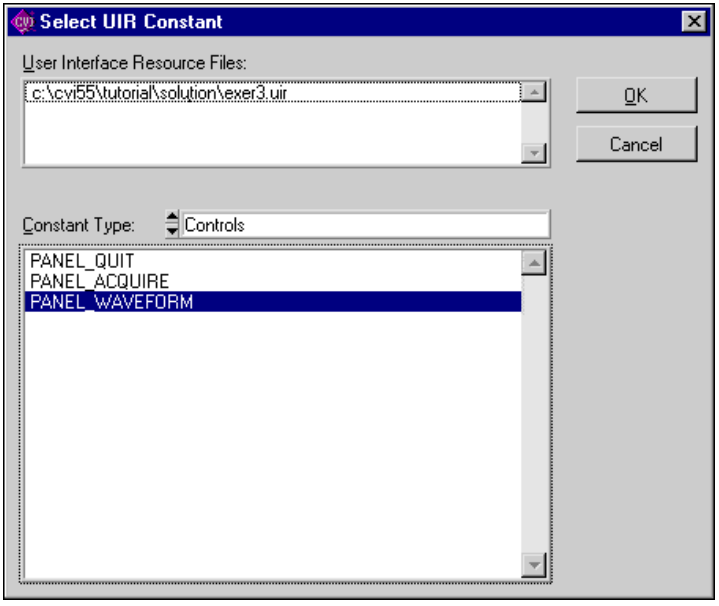

Figure 6-5. Select UIR Constant Dialog Box

Select `PANEL_WAVEFORM` and click **OK** to continue.

3. Type `datapoints` into the Y Array control. This name indicates which array in memory displays on the graph.

4. Type `100` into the **Number of Points** control. This number indicates how many elements in the array to plot. When your `PlotY` function panel matches the one in Figure 6-6, you are ready to go to the next step.

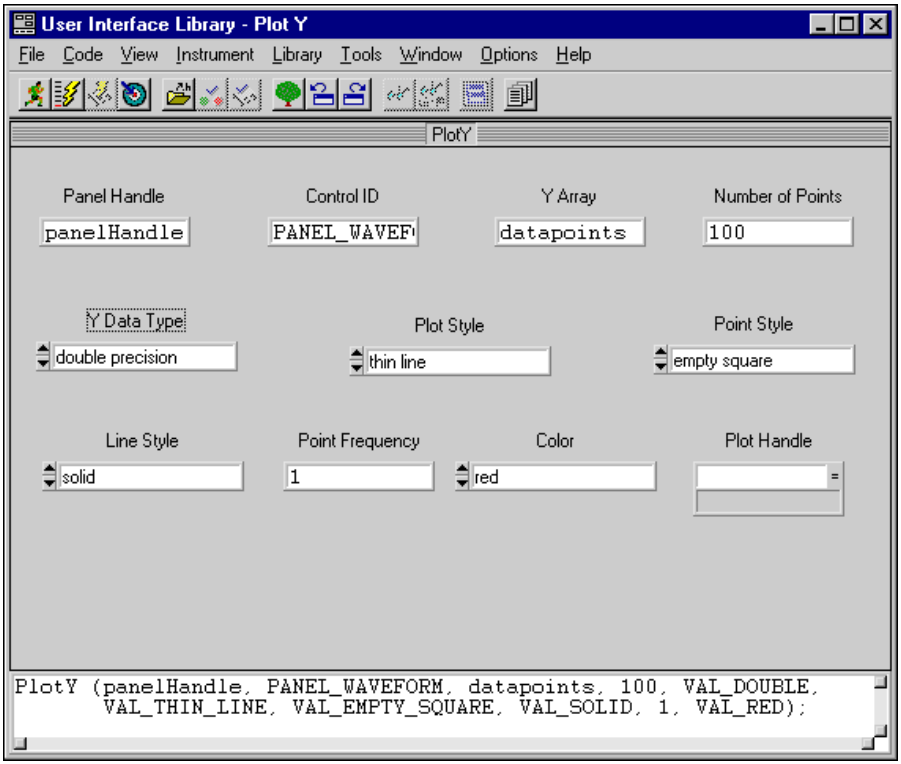

Figure 6-6. Completed Plot Y Function Panel

5. Select **Code»Set Target File** to indicate the window in which the function call is to be pasted. Select `sample4.c` (`cvi\tutorial\sample4.c`) from the dialog box and press <Enter> or click **OK**.
6. Select **Code»Insert Function Call** to paste the `PlotY` function call into your source code.

7. Close the function panel by selecting **File»Close**.

   Confirm that your program matches the source code shown in Figure 6-7.

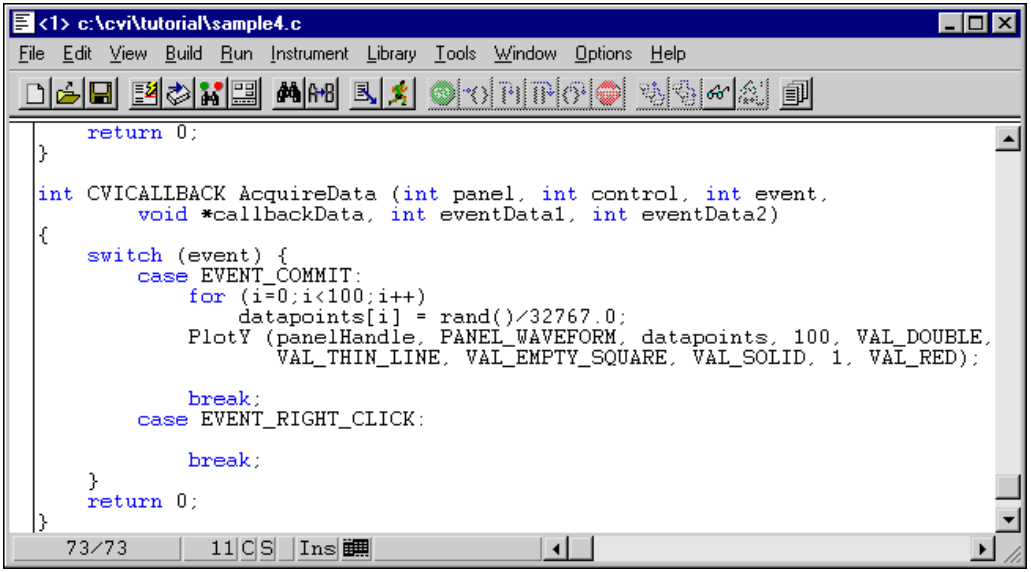

Figure 6-7. Completed Source Code for sample4.c

8. Select **File»Save**, enter `sample4.c` in the dialog box and press <Enter>.

Constructing the Project

In the first part of Chapter 2, *Loading, Running, and Editing Source Code*, you learned about the Project window. The Project window contains a list of files that make up the program you are developing. In this case, the following three files are listed in your Project window.

- `sample4.uir`—The user interface file that you built in Chapter 5,*Building a Graphical User Interface*.
- `sample4.h`—The include file that was generated automatically when you saved the `.uir` file in the chapter 5. The include file declares all the constant names and callback functions that you assigned in the User Interface Editor.
- `sample4.c`—The source file created in this chapter.

Follow these steps to construct your project.

1. Close all windows except the Project window (`untitled1.prj`).
2. Select **Edit»Add File to Project»All Files (*.*)**. This choice displays a dialog box that lists all the files in the `tutorial` directory that you are adding to your project list.

3. Select `sample4.c`, `sample4.h`, and `sample4.uir` from the dialog box by double-clicking on each file. The filenames appear in the Selected Files section of the dialog box shown in Figure 6-8.

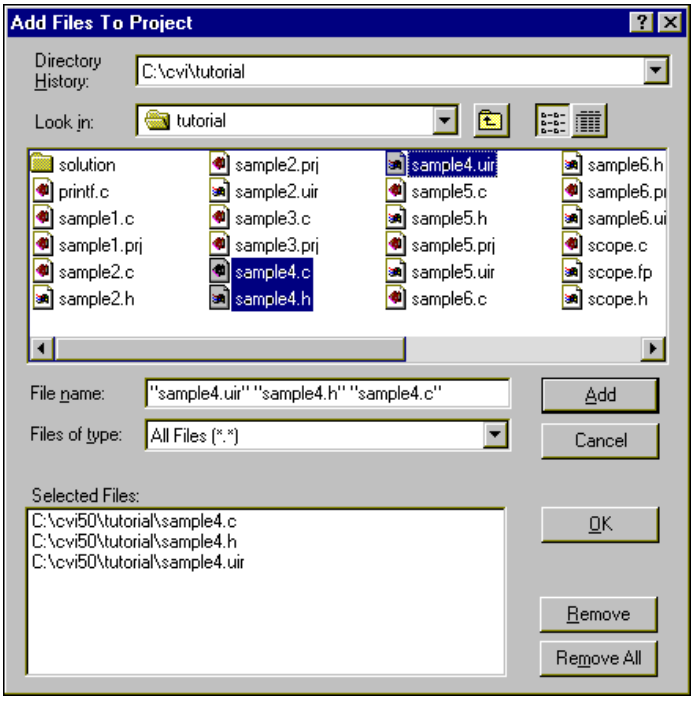

Figure 6-8. Add Files to Project Dialog Box

4. Click **OK**.
5. Select **File»Save**, enter `sample4` in the **File Name** control and press <Enter> or click **Save.**

Running the Completed Project

You now have a completed project, saved as `sample4.prj`. You can view the status of each file associated with the project in the Project window and edit each file by double-clicking the filename in the project list. Select **Run»Debug sample4_dbg.exe** to execute the code.

During the compile process, LabWindows/CVI recognizes that your program is missing the `ansi_c.h` include statement. Click **Yes** to add this include file to your program. When prompted, save the changes to the `sample4.c` file before running. While your program executes, 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. The user interface is displayed, 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 is passed directly to the `Shutdown` function, which halts the program.

You have completed the activities for this chapter. In the Chapter 7, *Adding Analysis to Your Program*, you learn to add simple analysis capability to your program.

7

Adding Analysis to Your Program

In Chapter 6, *Using Function Panels and the 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 the Acquire button. In this chapter, you add a simple analysis code that computes the maximum and minimum values of the random array you generate. To do this, you 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.

The objectives of this chapter are as follows.

- Review how to add controls to a user interface resource (`.uir`) file in the User Interface Editor.
- Learn how to write a callback function.
- Learn about the Analysis Library.
- Review how to generate source code using function panels.
- Learn how to send numeric values to user interface controls.

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

Setting Up

1. Close all windows except the Project window.
2. Select **File»Open»Project (*.prj)**. Load `sample5.prj`.
3. Select **Run»Execute sample_dbg.exe** to verify the operation of the program. The `sample5` file matches the project you completed in Chapter 6, *Using Function Panels and the Libraries*.

Goals of Section

In this section, you perform the following tasks.

1. Add a command button control to the user interface.
2. Add two numeric readout controls to the user interface.
3. Set up a callback function in the source code file to be triggered by the added command button.
4. Generate source code from the Analysis Library to find the maximum and minimum values of the random number array.
5. Generate source code to display these values in the added numeric readout controls.

Because you have completed previous sections of this tutorial regarding the User Interface Editor (Chapter 5, *Building a Graphical User Interface*) and code generation (Chapter 6, *Using Function Panels and the Libraries*), the instructions for performing these tasks are brief in this chapter.

Modifying the User Interface

Your first task is to modify the user interface that you built in Chapter 5, *Building a Graphical User Interface*. Follow these steps.

1. Open the source code by double-clicking on the `sample5.c` filename in the Project window. This code is similar to the resulting code from the previous example. Place your cursor at the end of the file. This might be unnecessary because CodeBuilder's default preference setting is to append generated code to the bottom 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, return to the Project window and double-click on the `sample5.uir` filename in the Project window to open the User Interface Editor. Your goal is to modify the `.uir` to match the user interface shown in Figure 7-1.

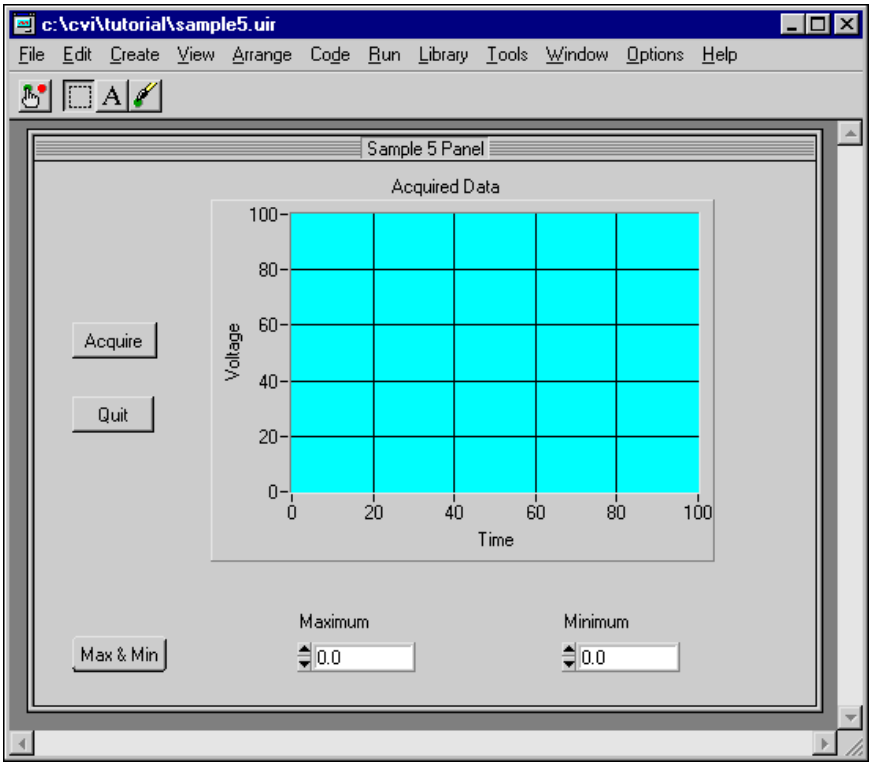

Figure 7-1. Sample User Interface

3. Select **Create»Command Button** and choose a command button from the control palette.
4. Double-click the **Command** button to invoke the Edit Command Button dialog box. Enter the following information into the dialog box.

   Constant Name: `MAXMIN`

   CallbackFunction: `FindMaxMin`

   Label: `Max & Min`

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

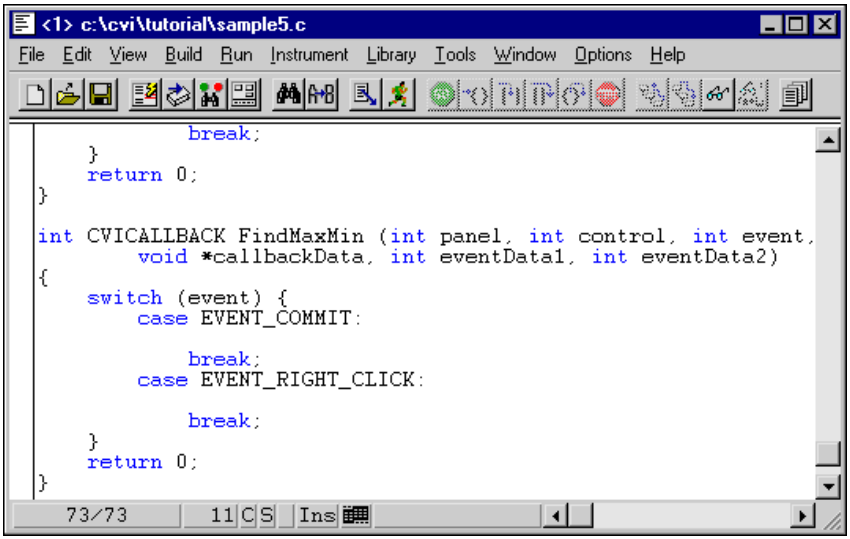

Figure 7-2. CodeBuilder—Generated Code

The lightning-bolt cursor appears while CodeBuilder generates code into the `sample5.c` source file as shown in Figure 7-2. When you finish updating the user interface for `sample5`, you add to the `FindMaxMin` callback function to compute and display the maximum and minimum values of the array.

In the User Interface Editor window, select **Create»Numeric** and choose the **Numeric** control from the upper left corner of the control palette, as shown in Figure 7-3.

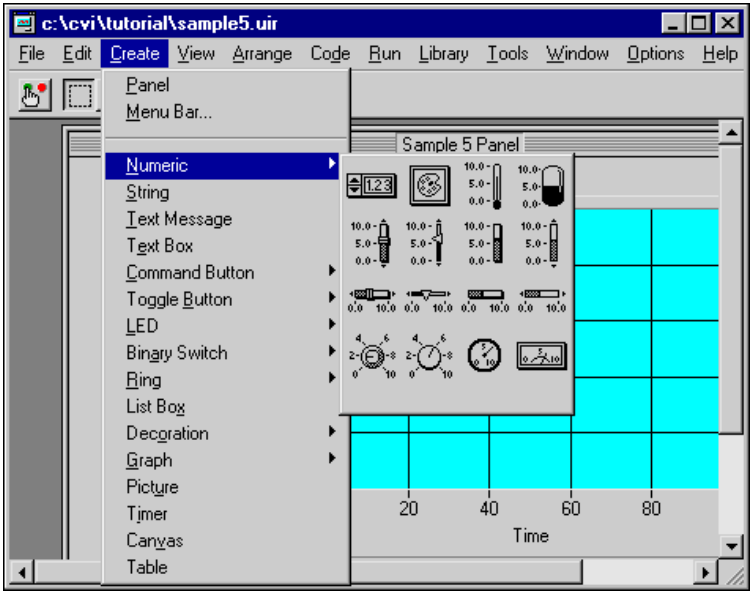

Figure 7-3. Selecting a Numeric Control from the Create Menu

6. Double-click on the **Numeric** control (labeled Untitled Control) to invoke the Edit Numeric dialog box. Enter the following information into the dialog box.

   Constant Name: `MAX`

   Control Mode: `Indicator` (Use the ring control.)

   Label: `Maximum`

7. Select **Create»Numeric** and choose the numeric control from the upper left corner of the control palette, as you did in Step 5.

8. Double-click the **Numeric** control to invoke the Edit Numeric dialog box. Enter the following information:

   Constant Name: `MIN`

   Control Mode: `Indicator` (Use the ring control.)

   Label: `Minimum`

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

10. Select **File»Save** to 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 compile the `FindMaxMin` function in the source file. Follow these steps.

1. You can use CodeBuilder to quickly locate the `FindMaxMin` callback function in your source file. Right-click the **Max & Min** button in the User Interface Editor to display the CodeBuilder pop-up menu. Select **View Control Callback**. CodeBuilder displays the `sample5.c` source file with the `FindMaxMin` callback function highlighted. The callback function appears as shown in Figure 7-4.

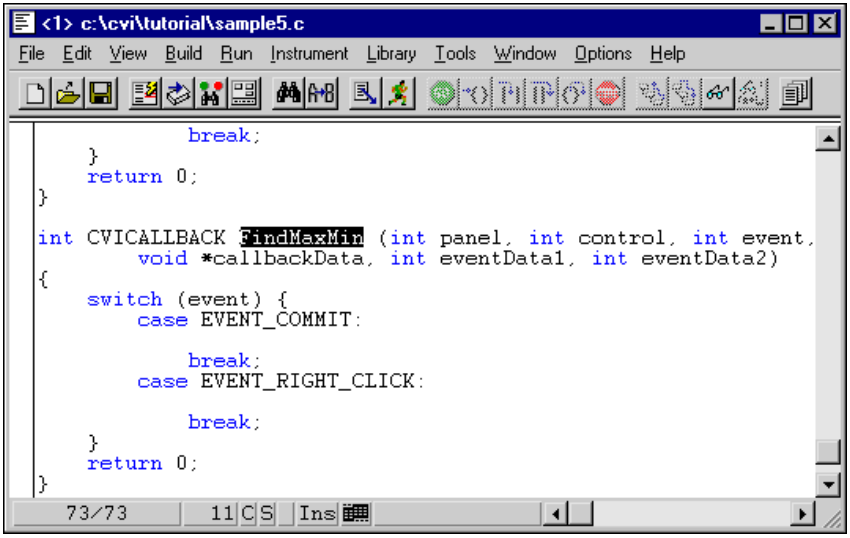

Figure 7-4. Using Code Builder to Find and Highlight the FindMaxMin Function

2. Position the input 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 into this area to find the maximum and minimum values of the datapoints array and display them on the user interface. You enter these function calls in the steps that follow.
3. Display the `Max & Min` function panel by selecting **Library»Analysis»Array Operations»1D Operations»1D Maximum & Minimum**.

Note Remember, depending on which package you have, your **Library** menu shows either the **Analysis** library or the **Advanced Analysis** library.

4. 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: datapoints

   Number of Elements: 100

   Maximum Value: max

   Maximum Index: max_index

   Minimum Value: min

   Minimum Index: min_index

5. Before you insert the MaxMin1D function into your source code, you must declare the variables max, max_index, min, and min_index. Click on the **Maximum Value** control to highlight it and select **Code»Declare Variable**. Set the checkbox to **Add declaration to current block**. This inserts a line of code to declare the max variable within the FindMaxMin callback function. Click **OK** to continue.
6. Notice that LabWindows/CVI automatically inserts an ampersand "&" before the max variable so that it is properly passed by reference to the function.
7. Repeat Step 5 for the **Maximum Index**, **Minimum Value**, and **Minimum Index** controls on the MaxMin1D function panel.
8. Insert the MaxMin1D function call into your source code by selecting **Code»Insert Function Call**. Close the MaxMin1D function panel. You see the MaxMin1D function inside the EVENT_COMMIT case statement within the FindMaxMin callback function.
9. Display the SetCtrlVal function panel by selecting **Library»User Interface» Controls/Graphs/Strip Charts»General Functions»Set Control Value**.
10. 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 display.

    Panel Handle: panelHandle

    Control ID: PANEL_MAX

    Value: max

11. Insert the SetCtrlVal function call into your source code by selecting **Code»Insert Function Call**. You then see the SetCtrlVal code entered on the line after the function call to FindMaxMin in your source code file.
12. Display the SetCtrlVal function panel again by selecting **Library»User Interface»Controls/Graphs/Strip Charts»General Functions»Set Control Value**.
13. 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

14. Insert the `SetCtrlVal` function call into your source code by selecting **Code»Insert Function Call**. Close the `SetCtrlVal` function panel.
15. Confirm that your source code matches the code shown in Figure 7-5.

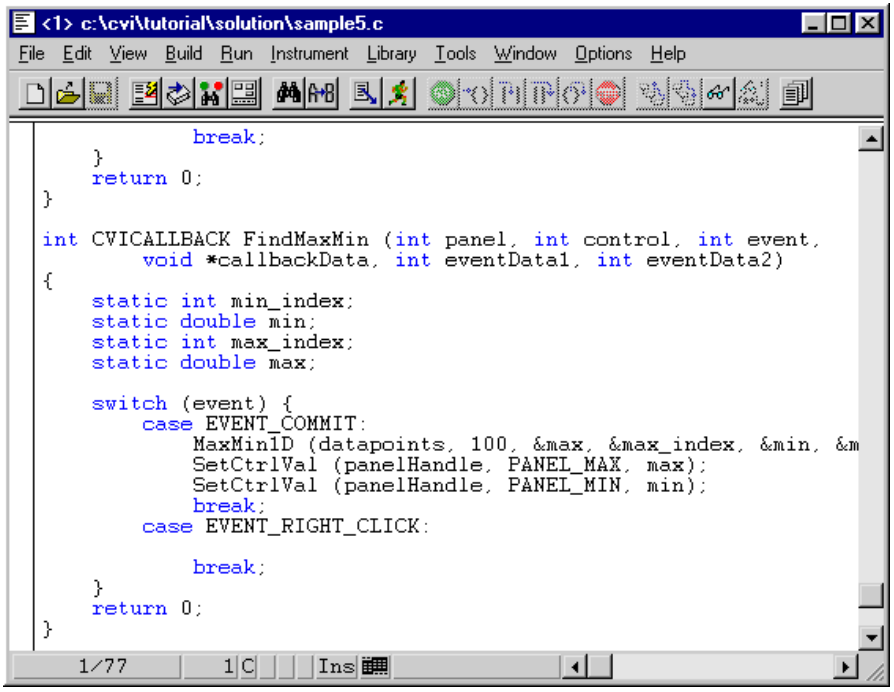

Figure 7-5. Completed Source Code for sample5.c

Running the Program

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

1. First, the button gets the input focus (`EVENT_GOT_FOCUS`).
2. Next, the down click of the left mouse button is sensed (`EVENT_LEFT_CLICK`).
3. Finally, the release of the left mouse button is sensed (`EVENT_COMMIT`). The `FindMaxMin` function is called for each event. You set the function to find the Minimum and Maximum values and display them only when you program receives the COMMIT event. For more practice with user interface events, complete exercise number 5 in Chapter 9, *Additional Exercises*.

Run the project. Remember, you must click **Acquire** first to generate the random data. Then you can click **Max & Min** to find the values.

This concludes this part of the tutorial. Close and save your file before moving on to the next chapter.

Chapter 8, *Using an Instrument Driver*, introduces you to the instrument drivers in LabWindows/CVI. You will use an instrument driver to acquire data instead of generating a random array.

8

Using an Instrument Driver

In this chapter of the tutorial you learn to use a simple instrument driver from the LabWindows/CVI Instrument Library. 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.

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, please do so now. Before beginning this example, follow these steps to set up the screen display.

1. Close all windows except for the Project window.
2. Select **File»Open»Project (*.prj)**.
3. Select `sample6.prj` from the dialog box.

Loading the Instrument Driver

An instrument driver consists of several files that reside on disk. Use the **Instrument** menu to load these files for use in LabWindows/CVI. Perform the following steps to load the sample instrument driver.

1. Select **Instrument»Load** in the Project window. The Load Instrument dialog box appears.
2. Press <Tab> repeatedly to select the large list box that contains directory and filenames or click anywhere inside the list box.
3. Select the `scope.fp` file from the `tutorial` directory. Press <Enter> to load the scope instrument driver.

4. Double-click on the `sample6.c` source file to display the code in a Source window.
5. Position the input cursor on the tagged line in the middle of the main program, the `DisplayPanel` function, as shown in Figure 8-1. You can move to this line quickly in the source code by pressing <F2>, which moves your cursor to the first tag set in the file.

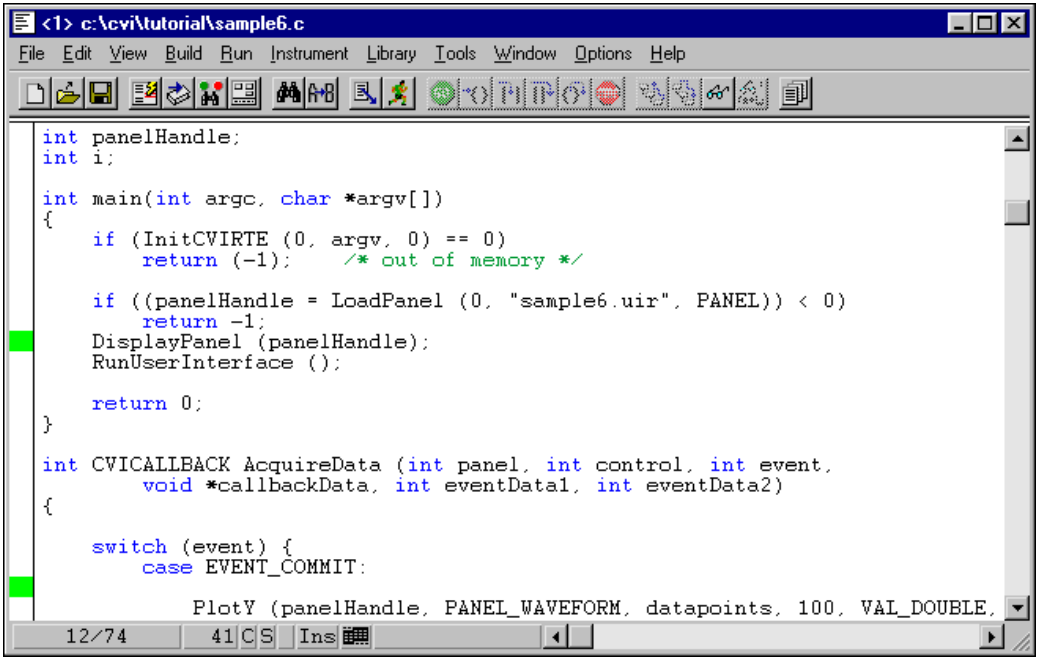

Figure 8-1. sample6.c Source File with Tags

To verify that the Scope instrument driver was loaded, display the **Instrument** menu. The **Sample Oscilloscope** item should appear on the menu as shown in Figure 8-2.

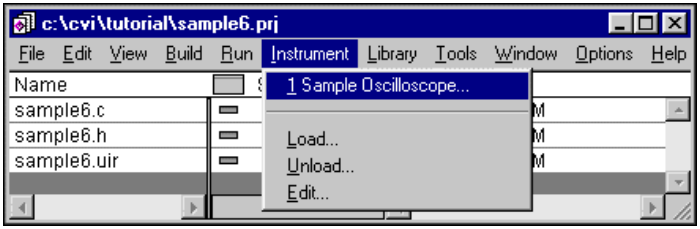

Figure 8-2. Sample Oscilloscope on Instrument Menu

Using the Instrument Driver

When the instrument driver is loaded, you can use it interactively in the same manner as the other LabWindows/CVI libraries through menus, dialog boxes, and function panels. Select **Instrument»Sample Oscilloscope**. The dialog box shown in Figure 8-3 appears.

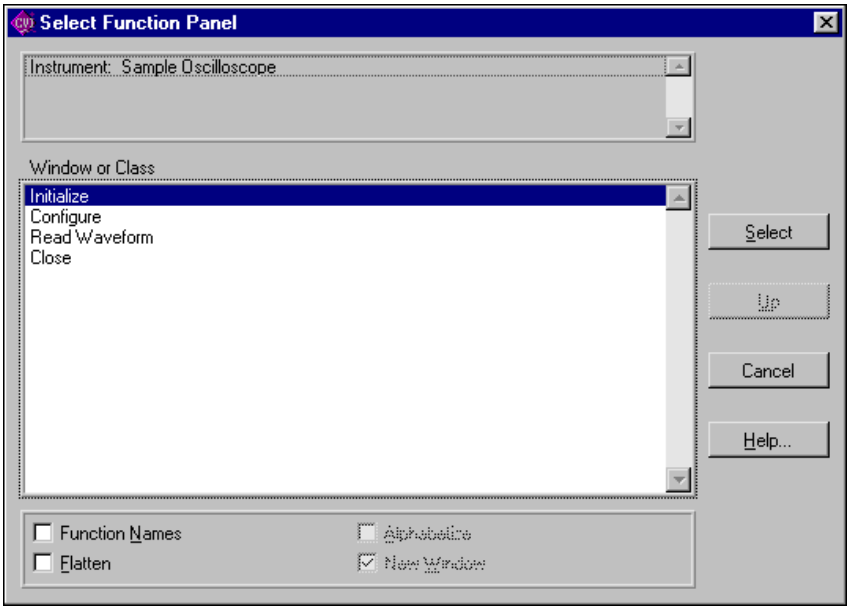

Figure 8-3. Sample Oscilloscope Functions

This module contains the functions

- `Initialize`,
- `Configure`,
- `Read Waveform`, and
- `Close`.

Use these functions to acquire a waveform in the sample program you are developing. Leave this dialog box open. Select a function after you read more about how to execute function panels before inserting code into your program.

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. Another important feature of function panels is the ability to execute the functions from the panel interactively, without writing a complete program. Therefore, you can experiment with functions by varying the parameter values at the panel and run them until you are satisfied with the result. Through trial and error, you can build your function calls at the function panel before inserting it into your source code. In this section you learn how to execute function panels before inserting the code into your program.

Initializing the Instrument

Each instrument driver uses a function to initialize the software and the instrument. You must execute the initialize function before using any other function in the module. Select **Initialize** from the dialog box. The function panel shown in Figure 8-4 appears.

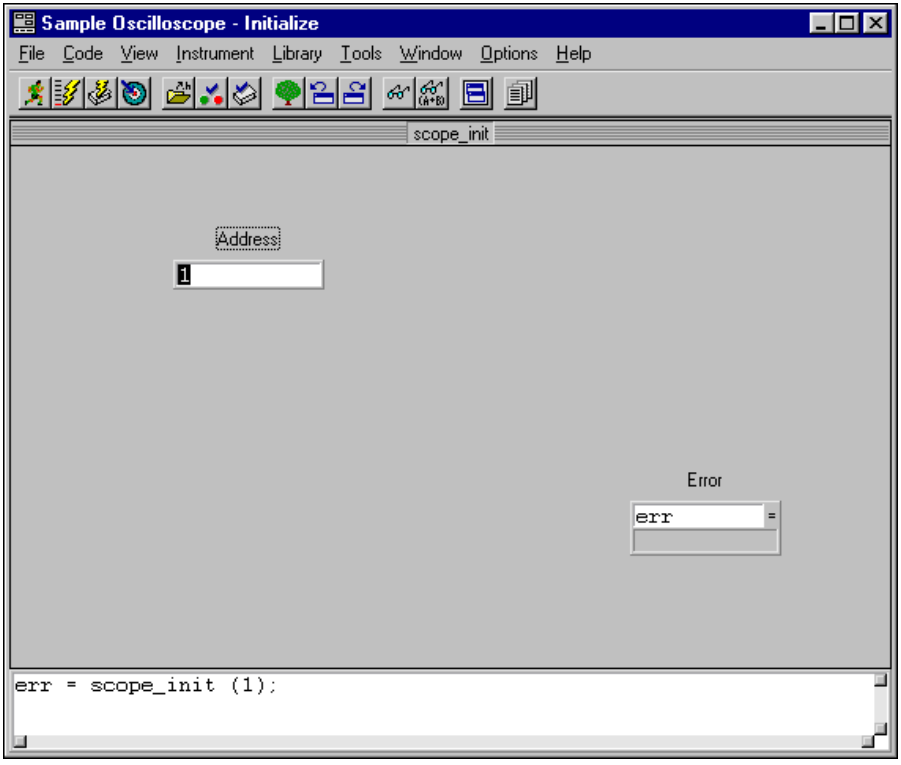

Figure 8-4. Initialize Function Panel

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

To get more information about the panel, select **Help»Window**. To view information about specific controls on the panel, select the desired control and press <F1> or click the secondary mouse button while on the desired control.

Follow these steps to initialize the instrument driver.

1. Enter the value `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 click the following checkbox items: **Execute declaration** and **Add declaration** to the top of target file. Click **OK**.
4. Select **Code»Run Function Panel**. Click **Yes** if a dialog box appears asking whether you want to save changes before running. This dialog box appears every time you make a change.

If no errors are detected during execution, the value in the Error control is 0. If the value is not 0, refer to the help information for the Error control to determine the cause of the problem.

Perform the following steps to copy the code to the Source window and remove the function panel from the screen.

1. Select **Code»Insert Function Call** or press <Ctrl-i> to copy the generated code to the Source window.
2. You might need to reset the target file for the inserted code. When you try to insert the code, the dialog box prompts you for the target file if you did not set it. Select `sample6.c` from the dialog box.
3. Select **File»Close**.

The function call to initialize the instrument driver appears above the `DisplayPanel` function call in the Source window as follows.

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

Configuring the Instrument

After the instrument is initialized, 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 `Configure` function. Select **Instrument»Sample Oscilloscope**. Select **Configure** from the dialog box. The function panel shown in Figure 8-5 appears on your screen.

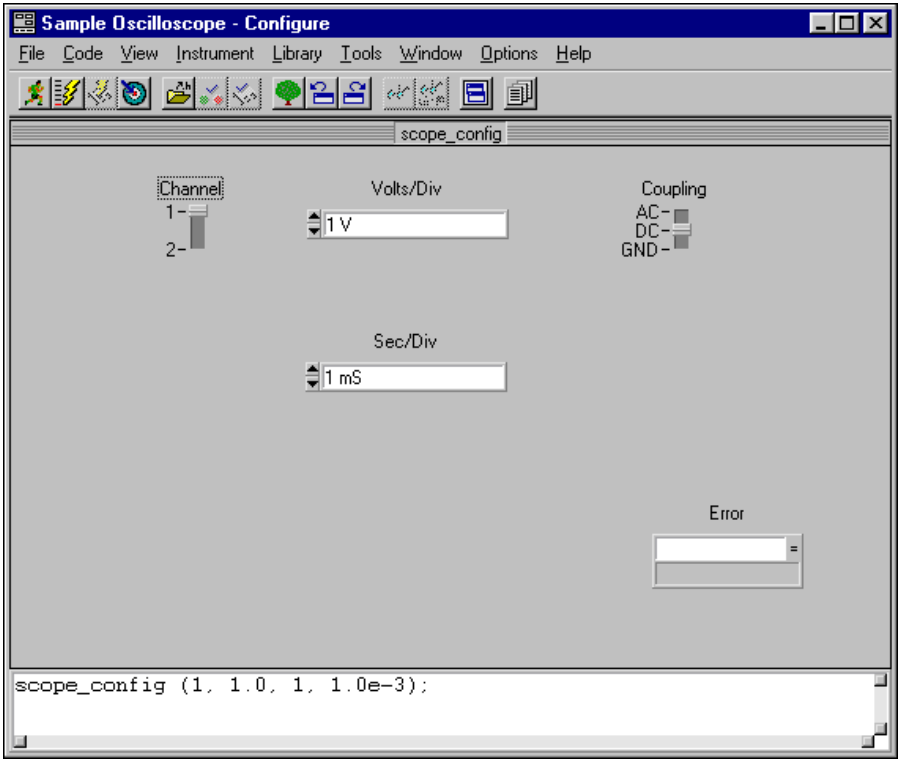

Figure 8-5. Configure Function Panel

The `Configure` 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 information for each control explains the purpose of the control and the valid range of inputs. Configure the panel the way you want it, keeping in mind that the way you configure the settings affects the waveform you read.

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

1. Enter `err` in the **Error** control.
2. Select **Code»Run Function Panel** to execute the function panel.

   If the Error control does not display a 0, an error has occurred. Refer to the help information to correct the problem and re-execute the function panel until the error is corrected.

   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.
3. Select **Code»Insert Function Call** to copy the generated code to the Source window. Close the function panel.
4. Move the input cursor to the blank line before the `PlotY` function in the `AcquireData` function. To do this quickly, press <F2>. 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.

Select **Instrument»Read Waveform»Sample Oscilloscope**. The Read Waveform function panel appears as shown in Figure 8-6.

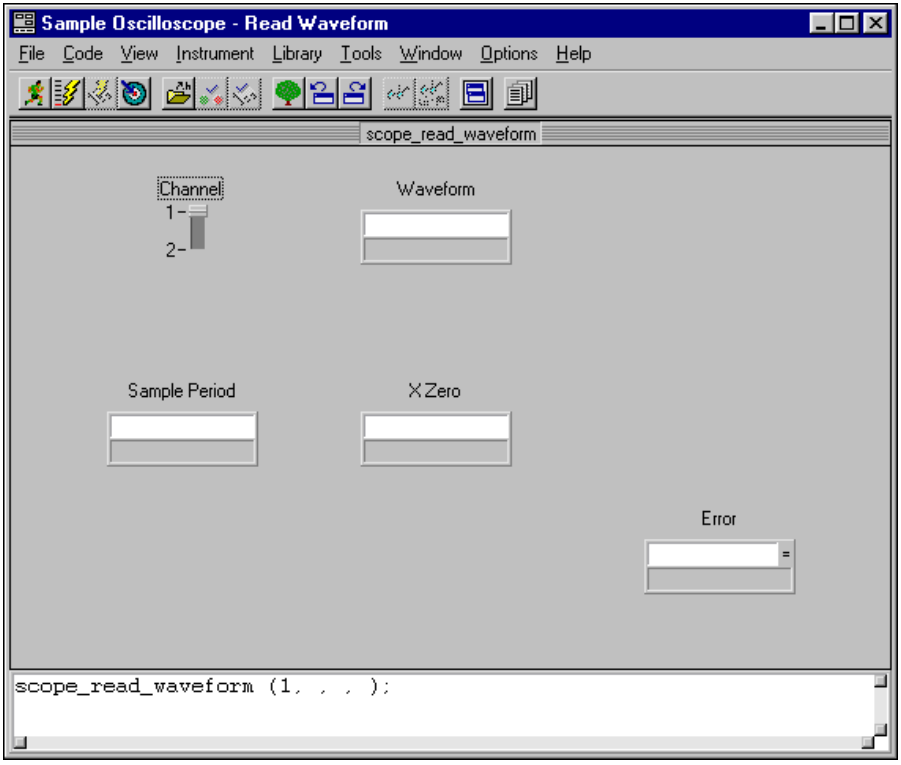

Figure 8-6. Read Waveform Function Panel

Set the Channel control to the channel you want to read. Channel 1 is a sinewave, and Channel 2 is random data.

Declaring Arrays from Function Panels

The `Read Waveform` function places the waveform data into an array. Before you can execute the function, you must declare an array for the waveform. You can declare variables, both scalars and arrays, from a function panel. To declare an array from the function panel, perform the following steps.

1. Press <Tab> to select the **Waveform** control or click on the label of the **Waveform** control.
2. Enter `datapoints` in the **Waveform** control.

3. In order to use the `datapoints` variable, you must first declare it in memory. Select **Code»Declare Variable**. A dialog box appears with the `datapoints` variable automatically entered in the **Variable Name** control.
4. Place your cursor in the **Number of Elements** text box and enter `100`.
5. Press <Tab> twice, so that the **Add declaration to the top of target file "sample6.c"** option is highlighted. Press <Space bar> to place a checkmark in the **Add declaration** checkbox, if it is not already present. Confirm that your Declare Variable dialog box matches the one shown in Figure 8-7.

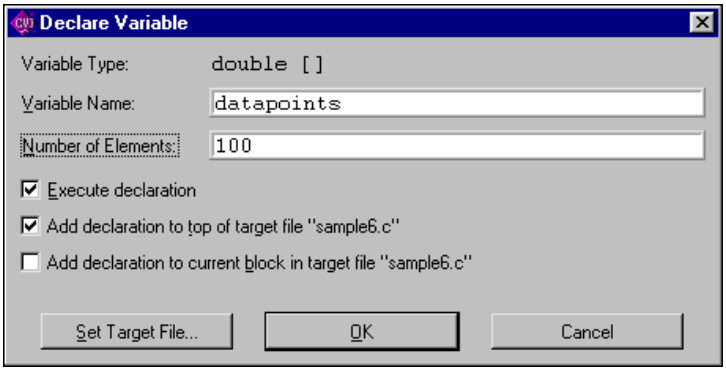

Figure 8-7. Declare Variable Dialog Box

6. Press <Enter> to declare the datapoints array.

Reading the Waveform

Complete the configuration of the function panel and run it as follows.

1. Click on the **Sample Period** control.
2. Select **Code»Declare Variable**.
3. Enter the variable name `delta_t` in the **Variable Name** input box and press <Enter>.
4. Press <Tab> to select the **X Zero** control.
5. Select **Code»Declare Variable**.
6. Enter the variable name `x_zero` in the **Variable Name** input box and press <Enter>.
7. Enter `err` in the **Error** control.
8. Select **Code»Run Function Panel** to execute the function panel. Save changes before running. If the Error control does not show 0, refer to the help information to correct the problem and run the panel again until a 0 appears. After the function has executed, a row of boxes in the Waveform control signifies that the data has been placed in the waveform array.

9. (Optional) To quickly view the data points acquired in the waveform array in the Variables Display, double-click on the row of boxes in the bottom half of the Waveform control on the function panel. Close the Variable Display.
10. Select **Code»Insert Function Call** to copy the generated code to the Source window.
11. Click the Source window in the background to view the source code that you just generated before the `PlotY` function.

Closing the Instrument

The last instrument-related operation performed is to close the instrument driver. Use this procedure to close the instrument driver.

1. In the Source window, position the cursor on the line in the `Shutdown` function with the following function call.

   ```
   QuitUserInterface(0);
   ```

   To do this quickly, press <F2>.
2. Select **Instrument»Sample Oscilloscope**.
3. Click **Close**. There are no parameters for the `Close` function panel. The `Close` function removes the instrument from a software configuration table. The instrument must be reinitialized before using it again.
4. Enter `err` in the **Error** control.
5. Select **Code»Run Function Panel** to close the instrument driver.
6. Select **Code»Insert Function Call** to copy the generated code to the Source window.
7. Click the Source window in the background to make it the active window.

Running the Program

The last step required before running the program is to include the scope header file. To call functions from the scope instrument driver, you must add the following line at the top of the source file.

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

Confirm that your program source code matches the following code.

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

#include <cvirte.h>
#include <analysis.h>
#include <ansi_c.h>
```

```
#include <userint.h>
#include "sample6.h"
#include "scope.h"

int panelHandle;
int i;
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 ();
    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 FindMaxMin (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    double max, min;
    int max_index, min_index;
    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;
}

int CVICALLBACK Shutdown (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            err = scope_close ();
            QuitUserInterface (0);
            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 illustration.

You are now ready to run the program. Select **Run»Debug sample6_dbg.exe**.

Adding the Instrument to Your Project

When you loaded the scope driver through the **Instrument** menu, you manually added the function panels to the instrument driver to LabWindows/CVI. By adding the scope driver to the file list in your project, the scope driver function panels are added to the **Instrument** menu automatically when you load the project in the future. Follow these steps to add the driver to your project.

1. Close all windows except the Project window.
2. Select **Edit»Add File to Project»Instrument (*.fp)**.
3. Select `scope.fp` from the dialog box, click **Add**, then click **OK**.

You have completed the activities for this chapter. In the Chapter 9, *Additional Exercises*, you work some additional exercises to practice what you learned in this part of the tutorial and to learn more about LabWindows/CVI.

9

Additional Exercises

This is the last chapter of the tutorial exercises. This chapter teaches you 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 `\cvi\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 `sample6` project that you completed in the 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, *Using an Instrument Driver*, exercises, start with the `sample6` project from the `solution` directory.

The `sample6` project generates a waveform and displays it on a graph control when a user clicks 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 appears in Figure 9-1.

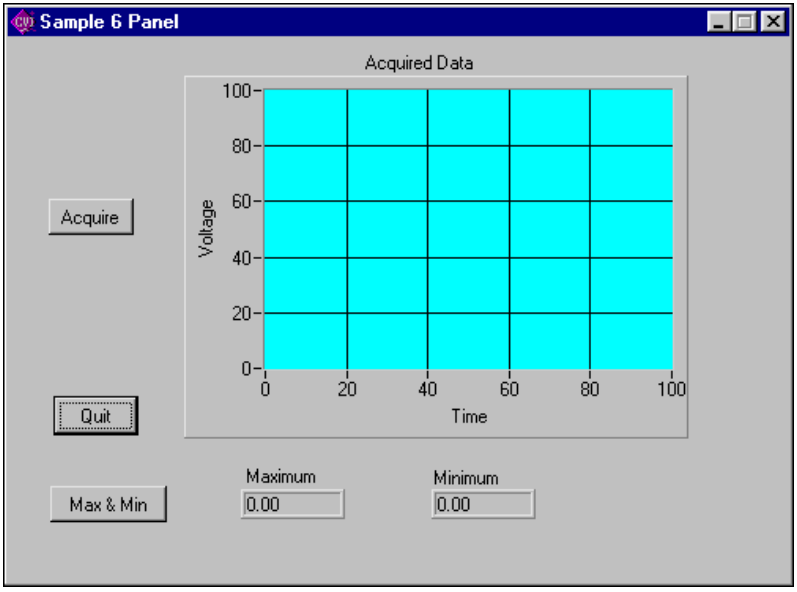

Figure 9-1. Sample User Interface

Exercise 1: Adding a Channel Control

Two of the most common functions you use in LabWindows/CVI are `SetCtrlVal` and `GetCtrlVal`. These functions are used to set and retrieve the current state of a control on a LabWindows/CVI `.uir` file. For example, you would 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, you would use `SetCtrlVal`. Both these functions take the same arguments as follows:

- A panel handle for the panel where the control exists
- The control ID for control to operate on
- A variable or value 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 to include a channel selection control, as shown in Figure 9-2 and modify the source code to properly acquire the correct channel.

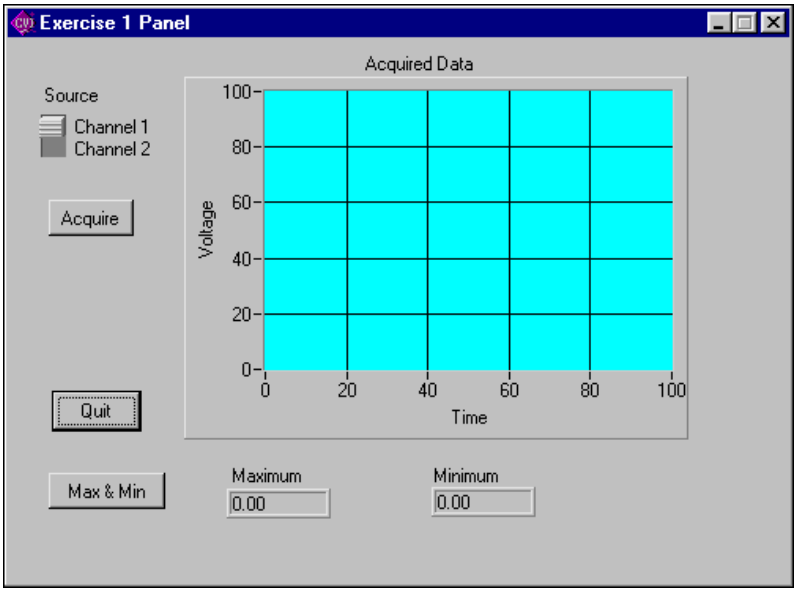

Figure 9-2. User Interface with Channel Selection Control

Hints

- Use a binary switch for the channel select 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 your 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 the label of a control in the User Interface Editor. These are all user interface control attributes.

You can 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 set and get 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 use these constants in the `GetCtrlAttribute` and `SetCtrlAttribute` functions.

Assignment

In this exercise, you 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, it is appropriate to 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 on 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 has 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. The Formatting and I/O Library was originally developed for the DOS version of LabWindows/CVI, and is included here for compatibility with existing programs.

Assignment

Use the file I/O functions in the ANSI C library to save the datapoints array to a text file in the `c:\cvi\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.

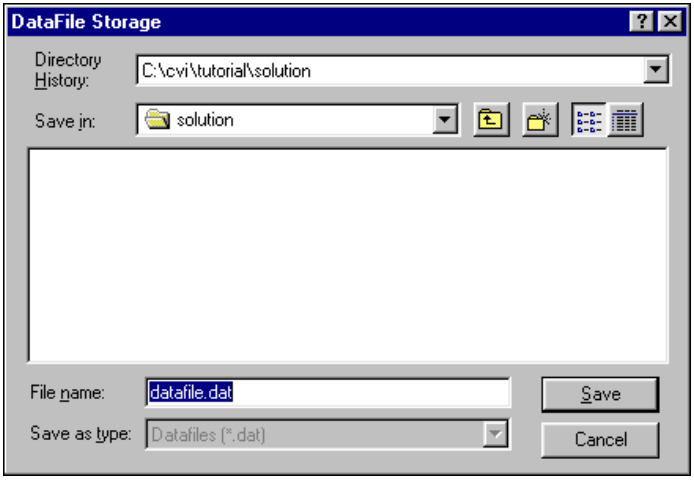

Figure 9-3. File Select Popup

Assignment

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

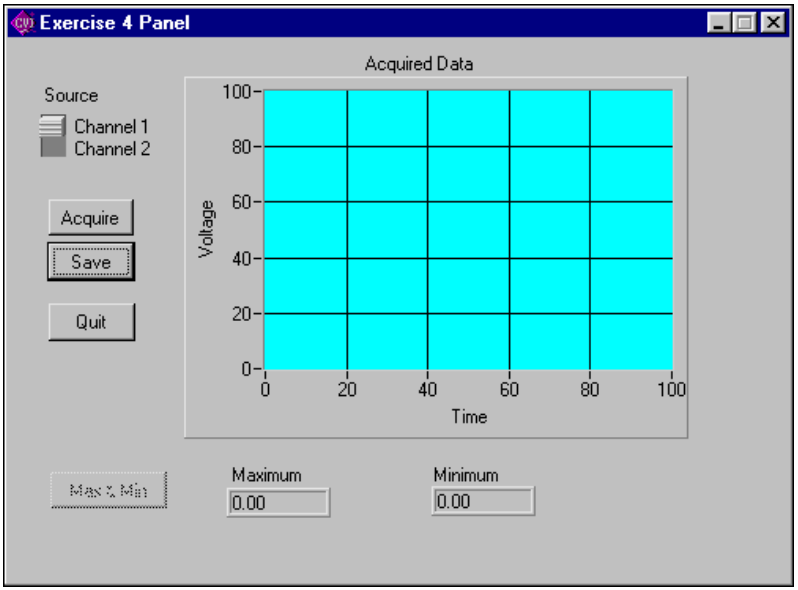

Figure 9-4. Completed User Interface

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 File Select Popup from the User Interface Library.

Solution: `EXER4.PRJ`

Exercise 5: 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 on a control.

User interface controls can generate many different types of events. For example, an event could be a left click or a right click. Or, an event could be a left double-click. In fact, events in LabWindows/CVI can be more than just mouse clicks. An event could 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.

```
EVENT_NONE
EVENT_COMMIT
EVENT_VAL_CHANGED
EVENT_IDLE
EVENT_LEFT_CLICK
EVENT_LEFT_DOUBLE_CLICK
EVENT_RIGHT_CLICK
EVENT_RIGHT_DOUBLE_CLICK
EVENT_KEYPRESS
EVENT_PANEL_MOVE
EVENT_PANEL_SIZE
EVENT_GOT_FOCUS
EVENT_LOST_FOCUS
EVENT_CLOSE
```

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 on 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_GOT_FOCUS` event if the button did not have the input focus before you clicked on it. The second time the callback function is called to process the `EVENT_LEFT_CLICK` event. The third time it is called to process the `EVENT_COMMIT` event. 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. Your GUI might be very easy to use, but usually it is preferable to add online help for the controls on your `.uir` panels to assist the operator. Alter `EXER4.PRJ` to pop up a short description for each command button when the user clicks on the button with the right mouse button.

Hints

- Use the `MessagePopup` function to display the help information.
- 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`.

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.

Timer Controls can be implemented into your program from the User Interface Editor. The Timer Control is visible only at design time in the User Interface Editor. At runtime, the timer control is not displayed. You can specify a constant name, callback function, and timer event interval in the Timer Control edit dialog box. LabWindows/CVI automatically calls the timer callback function specified 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 Editor 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 your Timer Control callback function.
- Use `SetCtrlVal` to display the random number on the thermometer.

Solution: `EXER6.PRJ`

You have now completed all the tutorial exercises in this manual. The remainder of this manual presents other information to help you get started using LabWindows/CVI.

Part III

Instrument Control, Data Acquisition, and LabWindows for DOS Conversions

- 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 for Windows. 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-DAQ software.
- Chapter 12, *Converting LabWindows for DOS Applications*, introduces the conversion tools in LabWindows/CVI for translating LabWindows for DOS applications into LabWindows/CVI applications. It also explains why certain LabWindows for DOS features are not supported in LabWindows/CVI.

10

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. The information included in this chapter is presented in more detail in the documentation that you receive with your hardware.

Getting Started with Your GPIB Controller

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

Introduction to GPIB

The General Purpose Interface Bus (GPIB) is a bus protocol for controlling standalone, 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 obey the IEEE 488.1, IEEE 488.2, and HS 488 specifications for controllers. The National Instruments IEEE 488.2 compatible TNT4882C and NAT4882 GPIB controller ASICs continue to improve and advance GPIB communication.

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 Your GPIB Interface Board

LabWindows/CVI works with the following National Instruments GPIB interfaces.

- PCI-GPIB (Plug-in PCI interface)
- AT-GPIB/TNT (PnP) and AT-GPIB (Plug-in 16-bit ISA interface)
- GPIB-PCII/IIA (Plug-in 8-bit PC/XT interface)
- MC-GPIB (Plug-in 16-bit Micro Channel interface)
- GPIB-485CT-A (External Serial-GPIB Controller)
- GPIB-1284CT (External Parallel-GPIB Controller)
- GPIB-232CT-A (External Serial-GPIB Controller)
- PCMCIA-GPIB (Plug-in card for PCMCIA Type II slots)
- GPIB-ENET (External Ethernet TCP/IP-GPIB Controller)

Each of these hardware kits comes with detailed information on how to configure and install your GPIB hardware.

Configuring Your 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 your controller. Some of these utilities include the following:

- 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

These and other utilities are described in the online documentation that you received with your GPIB software.

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. Do this by selecting **Options»Library Options...** in the Project window. Select the GPIB/GPIB 488.2 library option, as shown in Figure 10-1.

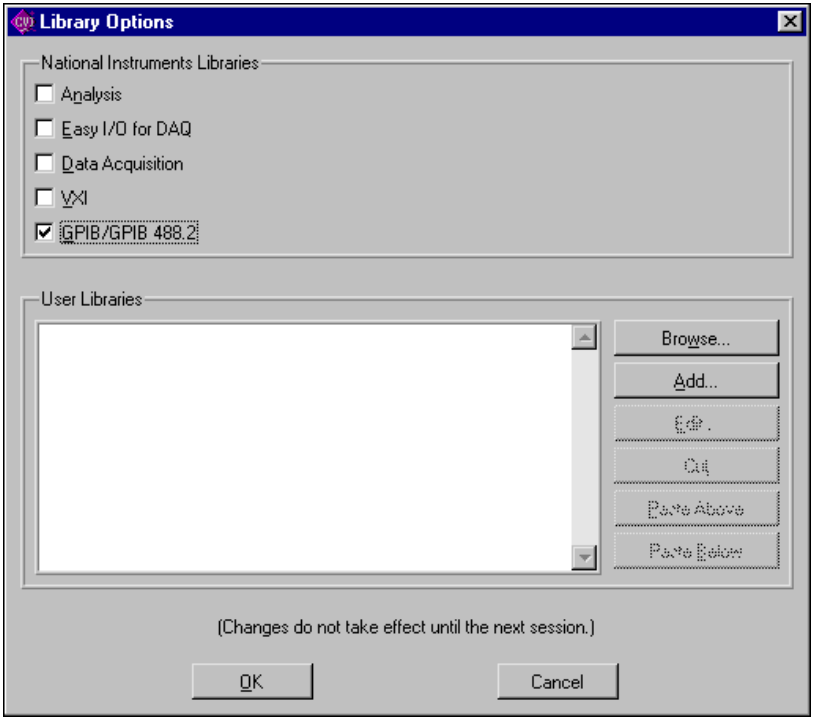

Figure 10-1. Library Options Dialog Box with GPIB/GPIB 488.2 Selected

Developing Your Application

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 came with your GPIB controller. This library is a DLL). While the function panels in LabWindows/CVI provide online help information for using these functions, detailed function descriptions for the GPIB 488/488.2 Library functions can be found in the online documentation that you receive with your GPIB software.

Getting Started with Your VXI Controller

The following sections include an introduction to VXI, information on the VXI Development system, and instructions for installing and configuring your VXI hardware, configuring your VXI software, developing your application, 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 and measurement and instrument control, and automated test equipment (ATE) applications. It is also 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 on 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 mentioned above.

Your 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 your VXI instruments.

Installing and Configuring Your VXI Hardware

LabWindows/CVI works with the following VXI controllers.

- VXIpc Model Series
- VXI-AT MXIbus Series
- VXI-PCI MXIbus Series
- GPIB-VXI/C

Each one of these controllers has documentation for installing and configuring the appropriate VXI hardware and software. For directions on how to install and configure your VXI hardware, refer to the getting started manuals for your controller.

Configuring Your VXI Driver Software

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

- **T&M Explorer**—A utility you can use to view your entire test and measurement system and configure various components.
- **NI-Spy**—A utility you can use to track the calls your application makes to National Instruments test and measurement drivers.
- **RESMAN**—The National Instruments multimainframe Resource Manager.
- **VIC**—Available as DOS executable. An interactive control program that executes VXI functions that you enter from the keyboard.

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. Do this by selecting **Options»Library Options...** in the Project window. Select the VXI library option, as shown in Figure 10-2.

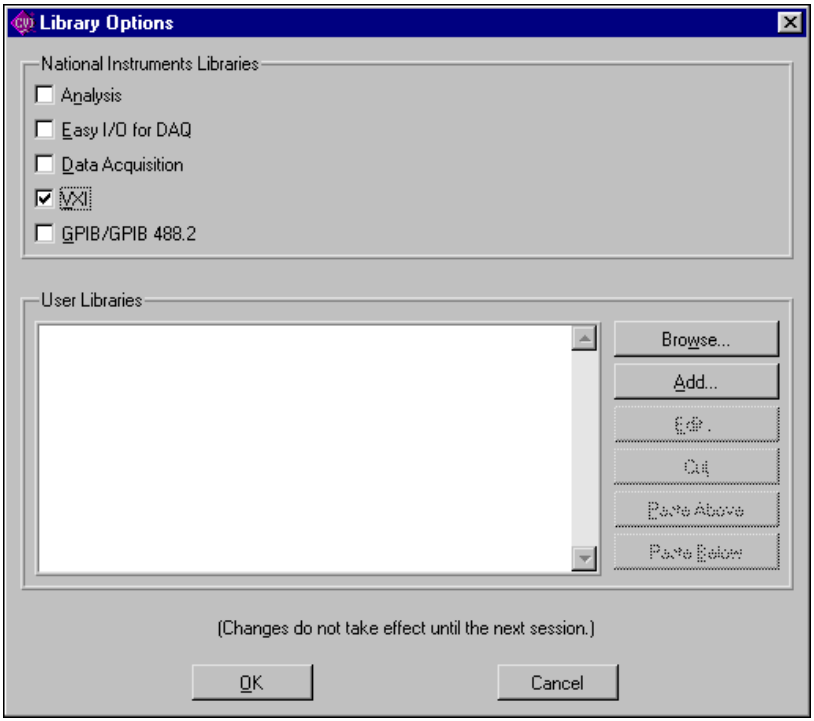

Figure 10-2. Library Options Dialog Box with VXI Selected

Developing Your Application

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 came with your VXI controller. This library is a DLL. The function panels in LabWindows/CVI provide some online help information for using these functions, but detailed function descriptions for the VXI Library functions are in the *NI-VXI Programmer Reference Manual*.

Using Instrument Drivers

Instrument control with LabWindows/CVI is simplified tremendously with the LabWindows/CVI Instrument Library. The Instrument Library contains drivers for hundreds of GPIB, serial, CAMAC, 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 your instrument is not part of the Instrument Library, you easily can convert an existing LabWindows/CVI instrument driver to control your instrument. To convert an existing driver, find an instrument driver in the same class, such as an oscilloscope, multimeter, or function generator, and change the commands in the source code to match your instrument.

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.

11

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. The information included in this chapter is presented in more detail in the documentation that you receive with your DAQ hardware and NI-DAQ 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-DAQ, is included with your DAQ device. LabWindows/CVI automatically loads the library of functions for controlling a National Instruments DAQ device if you have installed the NI-DAQ for Windows software on your PC.

Installing Your DAQ Device

The following section discusses the configuration issues for installing your DAQ device in your computer.

Configure Your Jumpers and DIP Switches

A DAQ device might have hardware switches or jumpers to configure the device for different modes of operation. Some devices have jumpers or switches for configuring the following settings.

- The *base I/O address* defines the I/O space for software communication with your device. The software writes to and reads from a number of registers on each device to control the device. Each of these registers has its own address that is an offset from the device base address. The device base address, therefore, defines the location of these registers. If two devices in your computer have the same base I/O address for any of their registers, a conflict occurs when the software tries to communicate with these devices.
- The *interrupt channel* designates which interrupt lines on the PC bus the device uses to assert interrupt signals. You can use interrupts to transfer data between the device and PC memory.
- The *direct memory access (DMA) channel* designates which DMA channels on the PC bus the device uses for transferring data. A DMA controller transfers data directly from the device to system memory without using the host computer CPU. You should therefore enable DMA jumpers for high-speed or background acquisition operations.

On some devices you also can configure the analog input range and analog input polarity and choose between single-ended or differential input mode.

Refer to your DAQ device documentation for more information on configuring your device.

Software Installation

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

After you have installed LabWindows/CVI, run the setup program on your NI-DAQ installation disks to install the NI-DAQ software. NI-DAQ accesses some directories that the LabWindows/CVI setup program creates. By selecting LabWindows/CVI from the NI-DAQ setup program, all the files that are required for DAQ operation are installed.

Configuring LabWindows/CVI for Data Acquisition

Before you can launch LabWindows/CVI and begin programming, you must first configure your DAQ device. Use the Measurement & Automation Explorer utility to configure your DAQ device. You must configure LabWindows/CVI to load the data acquisition libraries and associated function panels into the LabWindows/CVI programming environment. You do this by selecting **Options»Library Options** in the Project window. Select the Easy I/O for DAQ Library option and the Data Acquisition Library option, as shown in Figure 11-1.

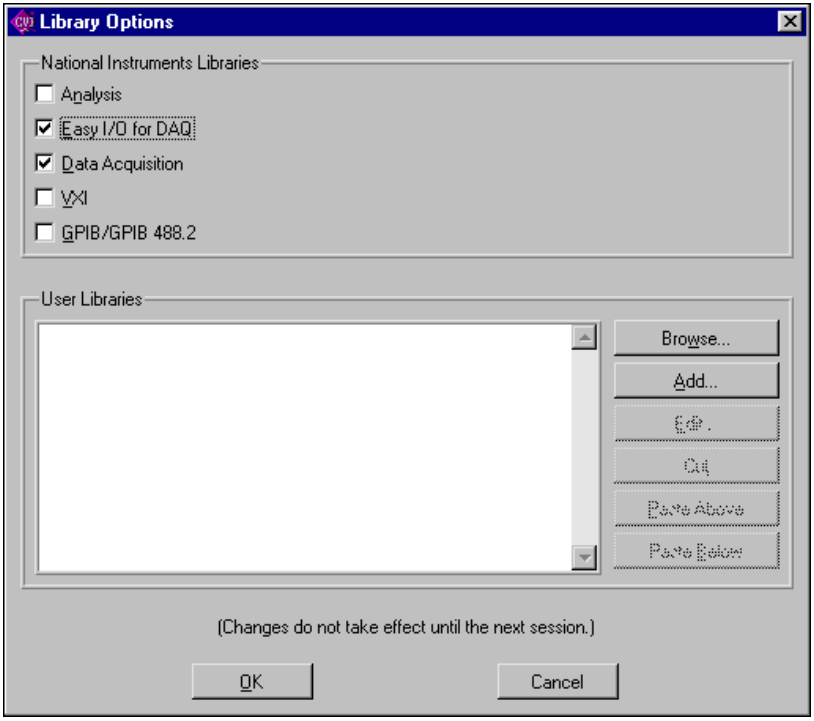

Figure 11-1. Library Options Dialog Box with Easy I/O for DAQ Selected

Test the Operation of Your Device and Configuration

At this point, you have configured and installed your DAQ device, installed LabWindows/CVI and the DAQ software, and configured your software with the NI-DAQ Configuration Utility. Before you start writing programs in LabWindows/CVI, it is a good idea to do some simple testing to make sure that your device is installed and functioning correctly. Two levels of simple testing exist for your DAQ device.

- **NI-DAQ Configuration Utility**—When you configured your system, you performed the first level of testing on your device. Options within the utility allow you to perform basic I/O operations with your DAQ device.
- **LabWindows/CVI Function Panels**—You can test the operation of your DAQ device from within LabWindows/CVI by interactively executing LabWindows/CVI function panels from the LabWindows/CVI Easy I/O for DAQ Library. Chapter 3, *Interactive Code Generation Tools*, explains how to use LabWindows/CVI function panels. You can execute the following LabWindows/CVI Easy I/O for DAQ Library functions interactively to test your device. After you load the function panel, select **Code»Run Function Panel**.
  - **AI Sample Channel**—The `AISampleChannel` function acquires a single voltage from a single analog input channel. You can connect a known voltage source to your DAQ device and verify that the analog input reading changes accordingly.
  - **AO Sample Channel**—The `AOSampleChannel` function applies a specified voltage to a single analog output channel.

For more information on how to execute LabWindows/CVI function panels, refer to Chapter 6, *Using Function Panels*, of the *LabWindows/CVI User Manual*.

Develop Your Application

When you know that your DAQ device is properly communicating with your software, you can begin developing your application. If you are new to LabWindows/CVI programming, Chapter 1, *Introduction to LabWindows/CVI*, contains a description of the event-driven programming that controls LabWindows/CVI graphical user interfaces. In addition, a set of custom controls with examples comes with LabWindows/CVI, and a set of example programs for performing common DAQ tasks comes with LabWindows/CVI and your NI-DAQ software when you install it with the LabWindows/CVI option. These tools and examples are a good starting point for your application.

Easy I/O for DAQ Library Sample Programs

The functions in the Easy I/O for DAQ Library make it easier to write simple DAQ programs than using the Data Acquisition Library. This library implements a high-level subset of the functionality of the Data Acquisition Library. For more information on this library and its functions, access function panel help for any of the functions in the Easy I/O for DAQ library. You access this library through the **Library** menu in LabWindows/CVI.

The sample programs for the Easy I/O for DAQ library are located in the `cvi\samples\easyio` directory.

Data Acquisition Library Sample Programs

The NI-DAQ for Windows software installs example programs into the `cvi\samples\daq` directory. Refer to the documentation in this directory for more details.

DAQ Control Instrument Drivers

The DAQ Control Instrument drivers make it easy to give DAQ functionality to certain user interface controls. The DAQ Control instrument drivers use the Easy I/O for DAQ Library.

DAQ Numeric Control Instrument Driver

The DAQ Numeric Control instrument driver implements numeric controls that have their values tied to analog input or output channels. The DAQ Numeric Control instrument driver is installed as `cvi\toolslib\custctrl\daq_num.fp` when you install LabWindows/CVI. A sample program for this instrument driver is in `cvi\samples\userint\custctrl\daq_num\daqndemo.prj`.

DAQ Chart Control Instrument Driver

The DAQ Chart Control instrument driver implements strip chart controls that can automatically scan a set of analog input channels at a specified rate and update the strip chart traces. You can configure the chart to check for alarm conditions and to keep a history buffer of the acquired data. The DAQ Chart Control instrument driver is installed as `cvi\toolslib\custctrl\daqchart.fp` when you install LabWindows/CVI. A sample program for this instrument driver is in `cvi\samples\userint\custctrl\daqchart\chartdem.prj`.

Event Function Parameter Data Types

Some parameters in the data acquisition event handling functions that are two bytes under Windows 3.1 increased in size to four bytes in subsequent versions of the Windows operating system. Typedefs have been added to the include file (`dataacq.h`) and the function panels so that you can successfully update your code. Table 11-1 shows the typedefs and the intrinsic types.

Table 11-1. Typedefs and Intrinsic Types for Different Platforms

Typedef	Windows 3.1	Windows NT/2000/98/95
DAQEventHandle	short	int
DAQEventMsg	short	int
DAQEventWParam	unsigned short	unsigned int
DAQEventLParam	unsigned long	unsigned long

The following function prototypes are affected by this change.

```
typedef void (*DAQEventCallbackPtr) (DAQEventHandle handle,
                            DAQEventMsg msg, DAQEventWParam wParam,
                            DAQEventLParam lParam);

short Config_Alarm_Deadband (short device, short mode,
                            char channelString[], double triggerLevel,
                            double deadbandWidth, DAQEventHandle
                            handle, DAQEventMsg alarmOnMessage,
                            DAQEventMsg alarmOffMessage,
                            DAQEventCallbackPtr EventFunction);

short Config_ATrig_Event_Message (short device, short mode,
                            char channelString[], double triggerLevel,
                            double windowSize, short triggerSlope,
                            long triggerSkipCount,
                            unsigned long preTriggerScans,
                            unsigned long postTriggerScans,
                            DAQEventHandle handle,
                            DAQEventMsg message,
                            DAQEventCallbackPtr eventFunction);
```

```
short Config_DAQ_Event_Message (short board, short mode,
                            char channelString[], short DAQEvent,
                            unsigned long triggerValue0,
                            unsigned long triggerValue1,
                            long triggerSkipCount,
                            unsigned long preTriggerScans,
                            unsigned long postTriggerScans,
                            DAQEventHandle handle,
                            DAQEventMsg message,
                            DAQEventCallbackPtr eventFunction);

short Get_DAQ_Event (unsigned long timeOut, DAQEventHandle *handle,
                            DAQEventMsg *message,
                            DAQEventWParam *wParam,
                            DAQEventLParam *lParam);

short Peek_DAQ_Event (unsigned long timeOut, DAQEventHandle *handle,
                            DAQEventMsg *message,
                            DAQEventWParam *wParam,
                            DAQEventLParam *lParam);
```

Source Code Changes Needed

If you wrote source code for Windows 3.1 that uses these functions and you want to use the source code under Windows NT/2000/98/95, you must modify your source code.

You must change the parameter declarations for all your event callback functions to match the new callback function prototype. Also, you must use the new typedefs in the declarations of variables that are passed by reference to `Get_DAQ_Event` and `Peek_DAQ_Event`.

Related Documentation

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

- *NI-DAQ User Manual for PC Compatibles*
- *NI-DAQ Function Reference Manual for PC Compatibles*
- *DAQ Hardware Overview Guide*
- *LabWindows/CVI User Manual*

12

Converting LabWindows for DOS Applications

This chapter introduces the conversion tools in LabWindows/CVI for translating LabWindows for DOS applications into LabWindows/CVI applications. It also explains why certain LabWindows for DOS features are not supported in LabWindows/CVI.

Conversion Tools

You might need to use LabWindows for DOS to get your files into the proper form for conversion. Once in the proper form, the conversion tools necessary for translating LabWindows for DOS applications into LabWindows/CVI applications are integral to LabWindows/CVI.

LabWindows/CVI has the following conversion tools.

- The C source code translator converts LabWindows for DOS C source files so you can use them in LabWindows/CVI. Refer to the *Converting Source Code* section for details about converting source code.
- The User Interface Resource (`.uir`) file translator converts LabWindows for DOS `.uir` files so you can use them in LabWindows/CVI. Refer to the *Converting User Interface Resource (.uir) Files* section for details about converting `.uir` files.
- The function panel (`.fp`) file translator converts LabWindows for DOS `.fp` files so you can use them in LabWindows/CVI. Refer to the *Converting Instrument Drivers* section for details about converting LabWindows for DOS instrument drivers.

This chapter also contains a section on converting LabWindows for DOS loadable compiled modules called *Converting Loadable Compiled Modules and External Modules*.

Unsupported Features

The features of LabWindows for DOS that are not supported by LabWindows/CVI are as follows.

- **The BASIC Language**—In the LabWindows for DOS environment, you write programs using a subset of either BASIC or C. In the LabWindows/CVI environment, you write programs using full ANSI C. If you have a program written in the BASIC subset of LabWindows for DOS, you can use the Change Languages feature of LabWindows for DOS to translate your BASIC source code into C source code.
- **The Graphics Library**—Because the User Interface Library of LabWindows for DOS Version 2.*x* made the Graphics Library obsolete, the Graphics Library is not supported in LabWindows/CVI.
- **The AT-DSP2200 Library**—This library is not available in LabWindows/CVI.
- **Data Acquisition Library Features**—The Data Acquisition Library for Micro Channel PCs is not available in LabWindows/CVI. The Memory Management functions, such as `NI_DAQ_Mem_Alloc`, also are not available.
- **User Interface Library Features**—The `GetColorPaletteValue` and `SetColorPaletteValue` User Interface Library functions are not supported because colors are specified through an RGB value rather than through color palette manipulation. Specifying RGB values makes the support of True Color adapters possible. Plotter hardcopy output is not supported in LabWindows/CVI. Hardcopy output is restricted to graphics printers.
- **Utility Library Features**—The general purpose `PutKey` function is not supported in LabWindows/CVI. However, the `FakeKeystroke` function is provided for use in the User Interface Library.
- **RS-232 Library**—In LabWindows for DOS, the RS-232 Library errors are positive values. In LabWindows/CVI the errors are negative.

Functions with New Behaviors

This section includes functions from the User Interface Library and the Utility Library. Refer to the function descriptions for these libraries in the *LabWindows/CVI Online Help*.

User Interface Library

The `LoadMenuBar` and `LoadPanel` functions in LabWindows/CVI User Interface Library require a panel handle parameter that corresponds to the parent panel that contains the newly loaded menu bar or panel. Since LabWindows for DOS did not have parent panels, the function `DOSCompatWindow` can be used in place of the parent panel parameter for backward compatibility. The `DOSCompatWindow` function displays a window that serves the same function as the background screen of your LabWindows for DOS application.

The `DeletePlots` function is retained for backward compatibility, but the more flexible `DeleteGraphPlot` function is available in LabWindows/CVI for deleting individual graph plots.

The `DisplayPCXFile` function is retained for backward compatibility, but the more flexible `DisplayImageFile` function is available in LabWindows/CVI for dynamically displaying pictures as controls.

Utility Library

The `GetProjectDir` function replaces `GetProgramDir` function. The `GetKey` and `KeyHit` functions work only in the Standard Input/Output window, not in the User Interface panels. The `GetKey` function returns different key codes than in LabWindows for DOS, and it always activates the Standard Input/Output window. The `KeyHit` function always activates the Standard Input/Output window.

Converting User Interface Resource (.uir) Files

This section describes the procedure for converting your LabWindows for DOS `.uir` files so that you can use them in LabWindows/CVI.

To convert a `.uir` file, select **File»Open»User Interface (*.uir)**. LabWindows/CVI automatically converts the file and gives you the opportunity to save the converted file and rename the original file.

If the .uir file uses the LabWindows DOS System font, the dialog box in Figure 12-1 is displayed.

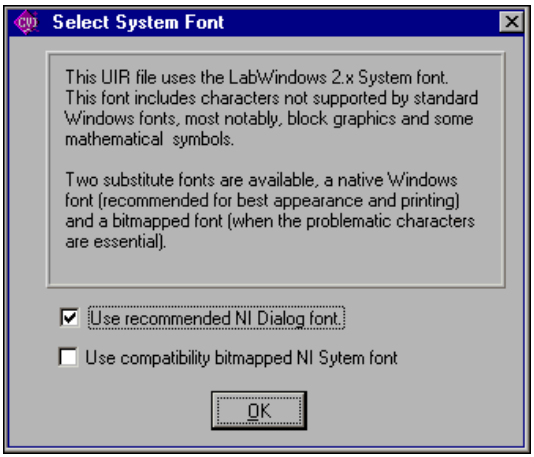

Figure 12-1. Select System Font Dialog Box

The LabWindows DOS System font contains special characters that are not supported by the default fonts of the host system. It is recommended that you use the NIDialog MetaFont in place of the System font. If the special characters are essential, you can use the compatibility-bitmapped NISystem font for the system font. However, strings that are in the compatibility-bitmapped NISystem font do not copy correctly between windows and applications.

Other fonts automatically convert to NIEditor or NIApp provided by LabWindows/CVI. You are free to change any text on the .uir to use any font supported on the host system, but only the *NI...* fonts are guaranteed to be on the PC.

For the most part, the .uir file appears as it did in LabWindows for DOS. In some cases you will have to make cosmetic changes manually. After you are satisfied with the appearance of the .uir file, save it so that your LabWindows/CVI program can use it.

Converting Source Code

This section describes the procedure for converting your LabWindows for DOS source code so it runs correctly in LabWindows/CVI.

To convert your C source code follow these steps.

1. Open any .uir files that your program uses. Refer to *Converting User Interface Resource (.uir) Files* section if you have not converted them yet.

2. Back up any source files you want to continue to use in LabWindows for DOS.
3. Open your source files in LabWindows/CVI by selecting **File»Open»Source (*.c)**.
4. Select **Options»Translate LW DOS Program**. This opens the dialog box shown in Figure 12-2.

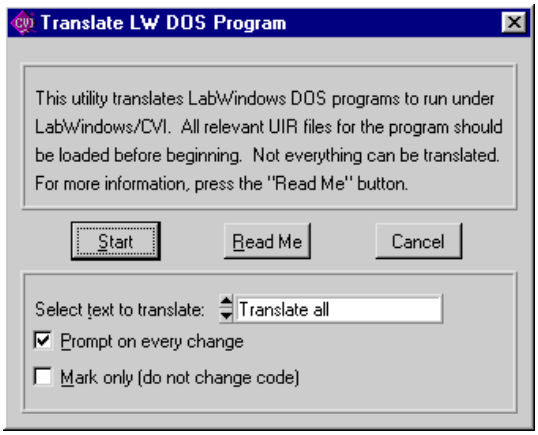

Figure 12-2. Translate LW DOS Program Dialog Box

- **Start** begins the translation process.
- **Read Me** displays a text description of the translation process.
- **Cancel** aborts the translation process.
- Select text to translate gives you the option of translating the entire file, translating from the current position of the cursor, or translating only the selected text.
- Prompt on every change invokes a dialog box each time a change to your source code is necessary. You can make, mark, or skip the change, undo the previous action, or stop the translation in this dialog box.
- Mark only adds comments to your program but does not make any functional changes.

5. After you configure the Translate LW DOS Program dialog box, click **Start** and complete the automated translation process.
6. The Analysis/Advanced Analysis Library functions do not update the global variable `dsp_err` in LabWindows/CVI. Instead, the return value for each function indicates the status of the function. If your program checks `dsp_err` for errors, you need to modify your program so that it checks the return values of the Analysis/Advanced Analysis Library functions for errors.
7. You need to modify any `int` arrays that are passed to GPIB, VXI, or DAQ functions. In LabWindows DOS, integer variables are 16 bit (2 bytes). In LabWindows/CVI, integer variables are 32 bit (4 bytes).

The contents of integer array elements 0 and 1 in LabWindows for DOS (a 16-bit system) are packed into integer array element 0 in LabWindows/CVI (a 32-bit system). Any attempt to access the array on an element-by-element basis will not work. Declare the array as `short` instead. Any type specifiers that refer to it should have the `[b2]` modifier when you pass them as an argument to a Formatting and I/O Library function.

8. You need to modify any `int` variables that you use in a way that requires them to be 2 byte integers.

   For example, if you pass an `int` argument by address to a function in the Formatting and I/O Library (a `Scan` source or a `Scan/Fmt` target) and it matches a `%d[b2]` or `%i[b2]` specifier, it will not work correctly. You need to remove the `[b2]` modifier or declare the variable as `short`.

   Conversely, if you pass a `short` argument by address and it matches a `%d` or `%i` specifier without the `[b2]` modifier, it will not work correctly. In this case, you must add the `[b2]` modifier.

Note Whereas the default for `%d` is 2 bytes on a 16-bit compiler, it is 4 bytes on a 32-bit compiler. Likewise, the default for `int` is 2 bytes on a 16-bit compiler, and it is 4 bytes on a 32-bit compiler. Thus, if the specifier for a variable of type `int` is `%d`, you do not need to make any modifications.

9. You need to modify any Formatting and I/O Library functions that use the `[o]` modifier. The `[o]` modifier in the Formatting and I/O Library has a different meaning in LabWindows/CVI than it does in LabWindows for DOS.

   In LabWindows for DOS, you use the `[o]` modifier to alter the ordering of the bytes that compose the integer. This is necessary if an instrument sends binary multibyte data and it is not in Intel format.

   Because LabWindows/CVI supports Intel (PC) architecture, you use the `[o]` modifier in LabWindows/CVI to *describe* the byte ordering in the data rather than to alter it. In a `Fmt/Scan` function, the buffer that contains the raw instrument data should have the `[o]` modifier describing the byte ordering. The buffer without the `[o]` modifier is guaranteed to be in the mode of the host processor. In other words, LabWindows/CVI reverses the byte ordering of the buffer without the `[o]` modifier depending on which architecture the program is running on.

   For example, if your GPIB instrument sends 2 byte binary data in Intel byte order, your code appears as follows.

```
short int instr_buf[100];
short int prog_buf[100];
status = ibrd (ud, instr_buf, 200);
Scan (instr_buf, "%100d[b2o01]>%100d", prog_buf);
```

The [o] modifier is used only on the buffer that contains the raw data from the instrument (instr_buf). LabWindows/CVI ensures that the program buffer (prog_buf) is in the proper byte order for the host processor. For a full description of the [o] modifier, refer to the discussion of the Formatting and I/O library in the *LabWindows/CVI Online Help*.

Converting Instrument Drivers

You can convert LabWindows for DOS instrument drivers for use in LabWindows/CVI. However, if National Instruments provided your LabWindows for DOS instrument drivers, you can acquire new LabWindows/CVI instrument drivers from the National Instruments Web site or ftp site. Refer to Appendix A, *Technical Support Resources*, for more information.

This section describes the procedure for converting your LabWindows for DOS instrument driver files so that you can use them in LabWindows/CVI.

1. Copy the existing *.c, *.h, and *.fp files for the instrument into a new directory to use in LabWindows/CVI.
2. Run LabWindows/CVI.
3. Create a new project. Add the *.c, *.h, and *.fp files for the instrument driver you want to port to the new project.

Convert the Instrument Driver Function Panels

1. Double-click the *.fp file in the project window to invoke the Function Tree Editor.
2. View every function panel in the instrument driver by selecting **Edit»Edit Function Panel Window**.
3. Spell out all extended ASCII characters. For example, “µ” becomes “micro.”
4. Size function panels appropriately on any Function Panel windows that have multiple function panels.
5. If you want to convert Function Panel help to the new style, select **Options»Help Style» New** from the Function Tree Editor window. Then select **Options»Transfer Window Help to Function Help**.
6. Save the *.fp file.

Convert the Instrument Driver Header File

1. To load the `*.h` file, double-click on it in the Project window.
2. Remove all instances of the `far` keyword.
3. Look for a global error variable declaration. If it exists:
   a. Use the `extern` keyword in the declaration. For example, `extern int tek_err;`.
   b. Define the error variable in the instrument driver source file. For example, `int tek_err;`.

   If there is not global error variable declaration in the header file, make sure it is declared as static in the instrument driver source file. For example, `static int tek_err;`.
4. Save the `*.h` file.

Convert the Instrument Driver Source Code

1. To load the `*.c` file, double-click on it in the Project window.
2. Continue with Step 3 in the *Converting Source Code* section in this chapter.

Converting Loadable Compiled Modules and External Modules

To convert a LabWindows for DOS loadable compiled module or external module, obtain the source code and follow the steps in the *Converting Source Code* section in this chapter.

Once the source code runs appropriately in LabWindows/CVI, you can compile it using LabWindows/CVI or a compatible external compiler. Refer to Chapter 2, *Using Loadable Compiled Modules*, in the *LabWindows/CVI Programmer Reference* for more details.

Note LabWindows for DOS loadable compiled modules or external modules that use DMA or interrupts that LabWindows/CVI does not support directly. You must rewrite these modules using the Microsoft DDK.

A

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.ni.com/support`

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.ni.com/worldwide`

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427,
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695,
Netherlands 0348 433466, Norway 32 27 73 00, Poland 48 22 528 94 06,
Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51,
Taiwan 02 2377 1200, United Kingdom 01635 523545

Glossary

Prefix	Meaning	Value
μ-	micro-	10^{-6}

A

active window — The window affected by keyboard input at a given moment. The title of an active window appears highlighted. Only an active window is visible in full-screen display mode.

Array display — A mechanism for viewing and editing numeric arrays.

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 `.uir` file to connect your GUI to the rest of your program. This code can be compiled and run as soon as it is created.

command button — A dialog box item that, when selected, executes a command associated with the dialog box.

control — An input or output device that appears on a function panel for specifying function parameters and displaying function results.

cursor	The flashing rectangle that shows where you may enter text on the screen. If you have a mouse installed, there is also a rectangular mouse cursor, or pointer.
cursor location indicator	An element of the LabWindows/CVI screen that specifies the row and column position of the cursor in the window.

D

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.
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

`.fp` 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 an instrument driver are grouped.

G

Generated Code box	A small window located at the bottom of the screen that displays the code produced by the manipulation of function panel controls.
global control	A function panel control that displays the value of a global variable within a function.

H

highlight — To make a LabWindows/CVI screen item ready for input.

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.

Instrument Library — A LabWindows/CVI library that contains instrument control functions.

Interactive execution window — A LabWindows/CVI work area in which sections of code can be executed without creating an entire program.

L

list box — A dialog box item that displays a list of possible choices for completing a command in the dialog box.

M

menu — An area accessible from the command 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.

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 window	A window that keeps track of the components that make up your current project. The Project window maintains a list of files such as source files, `.uir` files, header files, or object modules, and also contains status information about each file in your project.

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 complete programs are edited and executed. The file extension `.c` 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 Analysis, Formatting and I/O, GPIB, GPIB-488.2, RS-232, TCP, DDE 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.

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.

U

User Interface Editor	A graphical drag-and-drop editor for designing user interfaces for your program.
User Interface Editor window	This window 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

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

W

window	A working area that supports operations related to a specific task in the development and execution processes.
work area	The area of the LabWindows/CVI screen that contains the text displayed in a window.

Index

A

AcquireData function
    adding to user interface, 6-2
    assigning to Command button, 5-6 to 5-7
Add File to Project command, Edit menu
    adding instruments, 8-12
    constructing projects, 6-10
    illustration, 2-5
Add Watch Expression command, Options menu, 4-14
AISampleChannel function panel, 11-4
ANSI C Library, 1-3
ANSI C specifications, 2-8
AOSampleChannel function, 11-4
Array Display window
    displaying arrays, 4-11 to 4-12
    editing arrays, 4-12
arrays
    declaring from function panels, 8-8 to 8-9
    displaying, 4-11 to 4-12
    editing, 4-12
    generating random array of data, 6-3 to 6-4
AT-DSP2200 Library (LabWindows for DOS), 12-2
attributes, setting programmatically, 9-4

B

base I/O address for data acquisition boards, 11-2
BASIC programs, converting to C, 12-2
Break at First Statement command, Run menu, 4-2, 4-7, 4-13
breakpoints, 4-4 to 4-6
    breakpoint on error, 4-4
    conditional, 4-4
    definition, 4-4
    invoking, 4-4
    manual breakpoints, 4-4, 4-6
    programmatic breakpoints, 4-4 to 4-5
Build Error window, 2-6
Build Errors command, Window menu, 2-6
Build menu, Project window, 2-4

C

C language, using in LabWindows/CVI, 1-4
callback functions
    adding to Save button, 9-7
    adding with CodeBuilder, 7-4 to 7-5
    connecting source code with Command button, 5-6
    locating with CodeBuilder, 7-6 to 7-7
    processing events (example), 9-8 to 9-9
    specifying location for CodeBuilder generation, 7-2
    writing, 7-6 to 7-8
channel control, adding to project, 9-2 to 9-3
Chart Control Instrument Driver, DAQ, 11-5
Close command, File menu, 2-8
code. *See* source files.
code generation
    automatic. *See* CodeBuilder.
    interactive. *See* interactive code generation tools.
Code menu
    Function Panel windows
        Declare Variable command, 7-7, 8-9
        Insert Function Call command, 3-7, 6-9, 8-5, 8-7, 8-10
        Run Function Panel command, 8-5, 8-7, 8-10
        Select UIR Constant command, 6-8
        Set Target File command, 3-7, 6-9
    User Interface Editor, 5-11

CodeBuilder
adding control callback function, 7-4 to 7-5
generating program shell, 1-5, 5-10 to 5-13
locating FindMaxMin callback function, 7-6 to 7-7
Command button
adding for numeric control, 7-3 to 7-5
adding to user interface, 5-5 to 5-7
connecting with source code, 5-6 to 5-7
commit events, 7-9, 9-8
compiled modules. *See* loadable compiled modules.
compiler error messages, 2-6
conditional breakpoints, 4-4
configuration
data acquisition boards
jumpers and DIP switches, 11-2
LabWindows/CVI configuration, 11-3
testing, 11-4
GPIB driver software, 10-2
instrument drivers, 8-6 to 8-7
VXI driver software, 10-5
VXI hardware, 10-4 to 10-5
constants, displaying in .uir file, 6-8
Continue command, Run menu (table) , 4-3
Continue Execution command, Run menu, 4-14
Control Callback Events dialog box, 5-10
conventions used in manual, *xiii*
converting LabWindows for DOS applications
conversion tools, 12-1
functions with new behaviors, 12-3
User Interface Library, 12-3
Utility Library, 12-3
instrument drivers, 12-7 to 12-8
function panels, 12-7
header file, 12-8
source code, 12-8
loadable compiled modules or external modules, 12-8
source code, 12-4 to 12-7
unsupported features, 12-2
user interface resource (.uir) files, 12-3 to 12-4
Create menu
Command button, 7-3
Numeric command, 7-5
customizing toolbars, 2-8

D

DAQ boards. *See* data acquisition boards.
DAQ Chart Control Instrument Driver, 11-5
DAQ Control Instrument drivers, 11-5
DAQ Numeric Control Instrument Driver, 11-5
data
displaying and editing
arrays, 4-11 to 4-12
strings, 4-13
variables, 4-7 to 4-11
generating random array of data, 6-3 to 6-4
reading with instrument driver, 8-7 to 8-8
data acquisition
DAQ Control Instrument drivers, 11-5
developing applications, 11-4 to 11-5
Data Acquisition Library sample programs, 11-5
Easy I/O for DAQ Library sample programs, 11-5
event function parameter data type changes, 11-6 to 11-7
Windows 3.1 source code changes required, 11-7
overview, 1-6 to 1-7, 11-1
data acquisition boards
hardware configuration, 11-2

installation, 11-2
hardware installation, 11-2
software installation, 11-2
overview, 11-1
related documentation, 11-7
software configuration, 11-2
testing operation and configuration, 11-4
data acquisition libraries, 1-3
Data Acquisition Library
LabWindows for DOS, 12-2
sample programs, 11-5
data analysis
generating call to Mean function, 3-8
programming overview, 1-7
data analysis libraries, 1-3
data files, functions for reading and writing, 9-5
data types for event functions, changes in, 11-6 to 11-7
Debug command, Run menu
running completed project, 6-12
step mode execution, 4-3
debugging programs
breakpoints, 4-4 to 4-6
invoking, 4-4
manual breakpoints, 4-6
programmatic breakpoints, 4-4 to 4-5
displaying and editing data
Array Display, 4-11 to 4-12
String Display, 4-13
Variables window, 4-7 to 4-11
Watch window, 4-13 to 4-14
setting up tutorial for, 4-1 to 4-2
step mode execution, 4-2 to 4-3
Declare Variable command, Code menu
declaring arrays (example), 8-9
declaring variables (example), 7-7
reading waveform (example), 8-8 to 8-10
developing graphical user interfaces (GUI). *See* graphical user interface (GUI), building.
diagnostic resources, online, A-1
direct memory access (DMA) channel for data acquisition boards, 11-2
displaying and editing data, 4-6 to 4-14
arrays, 4-11 to 4-12
strings, 4-13
variables, 4-7 to 4-11
DisplayPanel function, 6-2
DMA channel for data acquisition boards, 11-2
documentation
conventions used in manual, *xiii*
data acquisition boards, 11-7
manuals for VXI controller boards, 10-5
related documentation, *xiii*

E

Easy I/O for DAQ Library sample programs, 11-5
Edit menu
Project window
Add File to Project command, 2-4 to 2-5, 6-10, 8-12
overview (table), 2-4
Source, Interactive Execution, and Standard Input/Output windows
editing tools, 2-8 to 2-9
Go To Definition command, 4-3
Undo command, 2-9
editing data
arrays, 4-12
strings, 4-13
variables, 4-9 to 4-11
editing tools in Source window, 2-8 to 2-9
error messages
closing error message window, 2-6
during compiling and linking, 2-6
Error window. *See also* Build Error window.
closing and opening, 2-6
EVENT_COMMIT, 7-9, 9-8
EVENT_GOT_FOCUS, 7-9, 9-8

EVENT_LEFT_CLICK, 7-9, 9-8
EVENT_RIGHT_CLICK, 5-10
events
  event function parameter data type changes, 11-6 to 11-7
  timed events, 9-10
  types of events, 9-8
example programs
  Data Acquisition Library sample programs, 11-5
  Easy I/O for DAQ Library sample programs, 11-5
  instrument drivers. *See* instrument driver programming example.
  NI-DAQ for Windows software, 11-5
external modules. *See also* loadable compiled modules.
  converting from LabWindows for DOS, 12-8

F

file I/O functions, ANSI C library, 9-5
File menu
  Project window
    Close command, 2-8
    Open command, 2-2
    overview (table), 2-4
  Source, Interactive Execution, and Standard Input/Output windows
    Open Quoted Text command, 2-8
  User Interface Editor, 5-4
File Select Popup, 9-6
Find Function Panel command, View menu, 3-8
finding functions, 3-8
  function definitions, 4-3
  using CodeBuilder, 7-6 to 7-7
FindMaxMin function, compiling in source file, 7-6 to 7-8
For Loop dialog box, 6-4
Function command, Help menu, 3-5
function panel controls
  generating events, 9-8 to 9-9
  output, 3-9
function panels
  adding YGraphPopup function to sample program, 3-7 to 3-8
  analyzing data, 3-8
  controls, 3-5
  converting instrument driver function panels, 12-7
  declaring arrays, 8-8 to 8-9
  definition, 3-4
  drawing a graph, 3-5 to 3-6
  executing functions interactively, 8-4
  finding functions, 3-8
    function definitions, 4-3
    PlotY function in User Interface Library, 6-4 to 6-7
    using CodeBuilder, 7-6 to 7-7
  Generated Code box, 3-7
  help information, 3-5 to 3-6
  input control, 3-5
  inserting code from function panels, 3-7 to 3-8
  output control, 3-9
  purpose and use, 3-4
  recalling, 3-10
  selecting, 6-4 to 6-7
  testing operation of DAQ device and configuration, 11-4
function trees (figure), 3-2

G

General Purpose Interface Bus (GPIB). *See* GPIB boards.
Generate All Code command, Code menu, 5-11
Generate All Code dialog box, 5-11

Generate Control Callback command, 7-4
Generated Code box, 3-7
generating code automatically. *See* CodeBuilder.
GetCtrlAttribute function, 9-4
GetCtrlVal function, 9-2 to 9-3
Go To Definition command, Edit menu, 4-3
GPIB boards
  compatible LabWindows/CVI interfaces, 10-2
  configuration
    GPIB driver software, 10-2
    LabWindows/CVI for GPIB, 10-3
  developing applications, 10-3
  installation, 10-2
  overview, 10-1
graph controls
  adding to user interface, 5-7 to 5-9
  finding PlotY function, 6-4 to 6-7
graphical user interface (GUI)
  building. *See* graphical user interface (GUI), building.
  components (figure), 1-5
  example of GUI, 2-10 to 2-11
  illustration, 2-10
  overview, 1-5
graphical user interface (GUI), building
  accessing User Interface Library, 3-3 to 3-4
  AcquireData function, 6-2
  adding Command Button, 5-5 to 5-7
  adding graph control, 5-7 to 5-9
  adding Save button, 9-6 to 9-7
  building PlotY function call syntax, 6-8 to 6-10
  building user interface resource (.uir) file, 5-3 to 5-13
  constructing the project, 6-10 to 6-11
  finding PlotY function, 6-4 to 6-7
  main function, 6-2
  modifying, 7-2 to 7-5
  running the program, 6-12, 7-9
  saving .uir files, 5-9
  setting attributes programmatically, 9-4
  setting up tutorial, 5-1 to 5-3
  Shutdown function, 6-3
  source code requirements, 5-3
  using instrument driver, 8-3 to 8-10
  waveform generation project (figure), 9-2
  writing callback function, 7-6 to 7-8
Graphics Library (LabWindows for DOS), 12-2
graphs, drawing with function panels, 3-5 to 3-6
GUI. *See* graphical user interface (GUI).

H

header files
  for instrument drivers, converting from LabWindows for DOS, 12-8
  for user interface, viewing, 5-9
help information, adding to command buttons, 9-9
Help menu
  Function Panel windows, 3-5
  Project window, 2-4

I

IEEE-488 standard (GPIB), 10-1
initializing instruments, 8-4 to 8-5
input controls, adding to function panel, 3-5
Insert Function Call command, Code menu
  copying instrument driver code, 8-5, 8-7, 8-10
  inserting code from function panel, 3-7
  PlotY function panel example, 6-9
installation
  data acquisition boards, 11-2
  GPIB boards, 10-2

LabWindows/CVI components (table), 1-1
NI-DAQ driver software, 11-2
VXI controllers, 10-4 to 10-5
instrument driver programming
example, 8-1 to 8-12
adding sample program to project files, 8-12
closing instrument drivers, 8-10
configuring instrument drivers, 8-6 to 8-7
declaring arrays from function panels, 8-8 to 8-9
functions in Sample Oscilloscope instrument, 8-3
initializing instrument drivers, 8-4 to 8-5
interactive function panel execution, 8-4
loading, 8-1 to 8-2
reading data, 8-7 to 8-8
reading waveform, 8-8 to 8-10
running the sample program, 8-10 to 8-12
Sample Oscilloscope program on Instruments menu, 8-2
instrument drivers
available drivers, 10-7
converting from LabWindows for DOS, 12-7 to 12-8
function panels, 12-7
header files, 12-8
source code, 12-8
DAQ Control Instrument drivers, 11-5
DAQ Chart Control Instrument Driver, 11-5
DAQ Numeric Control Instrument Driver, 11-5
definition, 10-7
use of Easy I/O for DAQ Library, 11-5
Instrument Library, 1-3
Instrument menu, 8-2
interactive code generation tools. *See also* CodeBuilder.
accessing User Interface Library, 3-3 to 3-4
analyzing data, 3-8
drawing graphs using function panels, 3-5 to 3-6
executing function panels interactively, 3-12
finishing the sample program, 3-10 to 3-11
function panel controls, 3-5
function panel help, 3-5 to 3-6
inserting code from function panel, 3-7 to 3-8
Library menu, 3-1 to 3-2
output values on function panels, 3-9
recalling function panels, 3-10
setting up for tutorial, 3-1
interprocess communication application libraries, 1-3
interrupt channel for data acquisition boards, 11-2

J

jumpers and switches for data acquisition boards, 11-2

L

label for Command button, changing, 5-7
LabWindows for DOS applications, converting. *See* converting LabWindows for DOS applications.
LabWindows/CVI. *See also* specific windows.
learning to use quickly, 1-2
libraries, 1-3
location of components (table), 1-1
setting up the tutorial, 2-2
starting, 2-2
system overview, 1-2 to 1-3
VXI development system, 10-4

Library menu
function tree (figure), 3-2
selecting libraries, 3-1 to 3-2
User Interface command, 6-4
Library Options command, Options menu
loading data acquisition libraries, 11-3
loading GPIB libraries, 10-3
loading VXI libraries, 10-5 to 10-6
Line command, View menu, 2-9
Line Numbers command, View menu, 2-8
linker error messages, 2-6
loadable compiled modules, converting from LabWindows for DOS, 12-9
loading projects, 2-2 to 2-5
LoadPanel function, 6-2

M

main function, 6-2
manual. *See* documentation.
manual breakpoints
invoking, 4-4
performing, 4-6
Max & Min command button
adding to user interface, 7-4 to 7-5
finding FindMaxMin callback function with CodeBuilder, 7-6
modifying with SetCtrlAttribute function, 9-4
Maximum Index control, 7-7
Maximum Value control, 7-7
MaxMin1D function, adding to source code, 7-6 to 7-7
Mean function, generating call to, 3-8
MessagePopup function, 9-9
Minimum Index control, 7-7
Minimum Value control, 7-7

N

National Instruments Web support, A-1 to A-2
networking, libraries for, 1-3
Next Tag command, View menu, 2-9
NI-488.2 DLL, 10-3
NI-488.2 Library, 10-3
NI-DAQ driver software
Data Acquisition Library sample programs, 11-5
installation, 11-2
NI-DAQ Configuration Utility, 11-4
NI-Spy utility, 10-5
NI-VXI library, 10-6
Numeric command, Create menu, 7-5
numeric controls
adding to user interface, 7-5
DAQ Numeric Control Instrument Driver, 11-5

O

online problem-solving and diagnostic resources, A-1
Open command, File menu, 2-2
Open Quoted Text command, File menu, 2-8
Open User Interface command, File menu, 5-4
Options menu, Project window
Add Watch Expression command, 4-14
Library Options command, 10-3, 10-5, 11-3
overview (table), 2-4
Toolbar command, 2-8
Translate LW DOS Program command, 12-5
Oscilloscope, sample. *See* instrument driver programming example.
output controls, adding to function panel, 3-9

P

PlotY function
  building call syntax, 6-8 to 6-10
  finding in User Interface Library, 6-4 to 6-7
  function panel (figure), 6-9
pop-up panels, 9-6 to 9-7
  adding to Save button, 9-6 to 9-7
  adding YGraphPopup to sample program, 3-7 to 3-8
  File Select Popup, 9-6
  purpose and use, 9-6
Preview User Interface Header File command, View menu, 5-9
problem-solving and diagnostic resources, online, A-1
program control
  components (figure), 1-5
  overview, 1-5 to 1-6
program development overview, 1-4 to 1-7. *See also* source files.
  program structure for LabWindows/CVI, 1-4 to 1-7
    data acquisition, 1-6 to 1-7
    data analysis, 1-7
    program control, 1-5 to 1-6
    program shell generation with CodeBuilder, 1-5
    relationship between elements (figure), 1-5
    user interface, 1-5
  using C in LabWindows/CVI, 1-4
program shell, building. *See* CodeBuilder.
programmatic breakpoints
  invoking, 4-4
  performing, 4-4 to 4-5
programming examples. *See* example programs.
programming graphical user interfaces. *See* graphical user interface (GUI), building.
programming tutorial. *See also* Project window; source files; Source window.
  additional exercises, 9-1 to 9-10
    adding channel control, 9-2 to 9-3
    setting user interface attributes programmatically, 9-4
    storing waveform on disk, 9-5
    timed events, 9-10
    user interface events, 9-8 to 9-9
    using pop-up panels, 9-6 to 9-7
  debugging programs
    breakpoints, 4-4 to 4-6
      manual breakpoints, 4-6
      programmatic breakpoints, 4-4 to 4-5
    displaying and editing data
      Array Display, 4-11 to 4-12
      String Display, 4-13
      Variables window, 4-7 to 4-11
      Watch window, 4-13 to 4-14
    step mode execution, 4-2 to 4-3
  editing tools, 2-8 to 2-9
  function panels
    adding YGraphPopup function to sample program, 3-7 to 3-8
    analyzing data, 3-8
    controls, 3-5
    declaring arrays, 8-8 to 8-9
    drawing graph, 3-5 to 3-6
    executing functions interactively, 8-4
    finding functions, 3-8
    finding PlotY function in User Interface Library, 6-4 to 6-7
    Generated Code box, 3-7
    help information, 3-5 to 3-6
    input control, 3-5
    inserting code from function panels, 3-7 to 3-8
    output control, 3-9
    recalling, 3-10

graphical user interface (GUI)
accessing User Interface Library, 3-3 to 3-4
AcquireData function, 6-2
adding Command Button, 5-5 to 5-7
adding graph control, 5-7 to 5-9
adding Save button, 9-6 to 9-7
analyzing source code, 6-1 to 6-3
building PlotY function call syntax, 6-8 to 6-10
building user interface resource (.uir) file, 5-3 to 5-13
constructing the project, 6-10 to 6-11
example of GUI, 2-10 to 2-11
finding PlotY function, 6-4 to 6-7
generating random array of data, 6-3 to 6-4
main function, 6-2
modifying, 7-2 to 7-5
running the program, 6-12, 7-9
saving .uir files, 5-9
setting attributes programmatically, 9-4
Shutdown function, 6-3
source code requirements, 5-3
waveform generation project (figure), 9-2
writing callback function, 7-6 to 7-8
instrument driver example, 8-1 to 8-12
adding sample program to project files, 8-12
closing, 8-10
configuring, 8-6 to 8-7
declaring arrays from function panels, 8-8 to 8-9
functions in Sample Oscilloscope instrument, 8-3
initializing, 8-4 to 8-5
interactive function panel execution, 8-4
interactive use, 8-3 to 8-4
loading, 8-1 to 8-2
reading data, 8-7 to 8-8
reading waveform, 8-8 to 8-10
running the sample program, 8-10 to 8-12
interactive code generation tools
accessing User Interface Library, 3-3 to 3-4
executing function panels interactively, 3-12
finishing the sample program, 3-10 to 3-11
function panel fundamentals, 3-4 to 3-10
Library menu, 3-1 to 3-2
setting up for tutorial, 3-1
Library menu, 3-1 to 3-2
loading projects, 2-2 to 2-5
running projects, 2-6
windows used for programming, 2-1 to 2-2
Project window
adding files to projects, 2-4 to 2-5
Build menu (table), 2-4
Edit menu (table), 2-4
File menu (table), 2-4
Help menu (table), 2-4
illustration, 2-1
information displayed about current project (figure), 2-5
loading projects, 2-2 to 2-5
Options menu (table), 2-4
Run menu, 2-6
Run menu (table), 2-4
title determined by current project, 2-1
Tools menu (table), 2-4
Transfer Project Options dialog box (figure), 5-2
Window menu (table), 2-4
projects. *See also* source files.

adding files to projects, 2-4 to 2-5, 6-10 to 6-11, 8-12
adding instruments, 8-12
error messages, 2-6
loading, 2-2 to 2-5
running
instrument driver example, 8-10 to 8-12
Max & Min command button sample, 7-9
sample1 project, 2-6
sample4.prj, 6-12

R

random array of data, generating, 6-3 to 6-4
Read Waveform function example, 8-9 to 8-10
reading data with instrument driver, 8-7 to 8-8
Recall Function Panel command, View menu, 3-10
RESMAN utility, 10-5
right-clicking on GUI control, 5-10
RS232 Library (LabWindows for DOS), 12-2
Run Function Panel command, Code menu, 8-5, 8-7, 8-9
Run menu
Interactive Execution window
Continue command, 4-3
Continue Execution command, 4-14
Debug command, 4-3, 6-12
Step Into command, 4-3, 4-5, 4-10
Step Over command, 4-3, 4-5
Terminate Execution command, 4-3, 4-5
View Variable Value command, 4-10
Project window
Break at First Statement command, 4-2, 4-7, 4-13
Debug command, 6-12
overview (table), 2-4
Run Project command, 2-6, 3-11
Run Project command, Run menu
running completed project, 3-11
sample1 project, 2-6
running projects
instrument driver example, 8-10 to 8-12
Max & Min command button sample, 7-9
sample1 project, 2-6
sample4.prj, 6-12
Runtime Errors command, Window menu, 2-6
RunUserInterface function, 6-2

S

Sample Oscilloscope program. *See* instrument driver programming example.
sample programs. *See* example programs.
Save button, adding to project, 9-6 to 9-7
scope.fp sample file, 8-1
Select UIR Constant command, Code menu, 6-8
Set Target File command, Code menu, 3-7, 6-9
SetCtrlAttribute function, 9-4
SetCtrlVal function
adding to source code, 7-7 to 7-8
purpose and use, 9-2
shells, building. *See* CodeBuilder.
Shutdown function, 6-3
software-related resources, A-2
solution subdirectory of tutorial directory, 9-1
source files. *See also* programming tutorial.
analyzing code, 6-1 to 6-3
connecting code with graphical user interface
assigning function to Command button, 5-6 to 5-7
requirements, 5-3
converting from LabWindows for DOS, 12-4 to 12-7
instrument drivers, 12-8

displaying in Source window, 2-6 to 2-8
displaying referenced files, 2-8
error messages, 2-6
information displayed in Project window (figure), 2-5
inserting code from function panels, 3-7 to 3-8
loading, 2-2 to 2-5
moving to specific lines of code, 2-9
opening subwindows for one source file, 2-8 to 2-9
recalling function panel for editing, 3-10
Source window
available menus, 2-7
compatibility with ANSI C specifications, 2-8
displaying generated code, 5-12
displaying source code, 2-6 to 2-8
editing tools, 2-8 to 2-9
illustration, 2-2, 2-7
opening subwindows, 2-8 to 2-9
sample3.c program (figure), 4-2
title determined by current source file, 2-1
Standard Input/Output window, 2-6
starting LabWindows/CVI, 2-2
Step Into command, Run menu, 4-3, 4-5, 4-10
step mode execution, 4-2 to 4-3
Step Over command, Run menu, 4-3, 4-5
String Display window
displaying and editing string variables, 4-13
illustration, 4-13
subwindows
illustration, 2-9
opening subwindows for one source file, 2-8

T

tagged lines, 2-9
technical support resources, A-1 to A-2
Terminate Execution command, Run menu, 4-3, 4-5
testing board configuration, 11-4
timed events, 9-10
Timer controls, 9-10
T&M Explorer utility, 10-5
Toggle Tag command, View menu, 2-9
Toolbar command, Options menu, 2-8
toolbars
customizing, 2-8
displaying names of button or icons, 2-8
Tools menu, Project window, 2-4
Transfer Project Options dialog box (figure), 5-2
Translate LW DOS Program command, Options menu, 12-5
Translate LW DOS Program dialog box, 12-5
typedefs, for event function parameter data types, 11-6 to 11-7

U

.uir files. *See* user interface resource (.uir) files.
Undo command, Edit menu, 2-9
user interface. *See* graphical user interface (GUI).
user interface, creating. *See* graphical user interface (GUI), building.
User Interface command, Library menu, 6-4
User Interface Editor window
illustration, 2-2
purpose and use, 5-2
title determined by current .uir file, 2-1
user interface events. *See* events.

User Interface Library
accessing, 3-3 to 3-4
contents, 1-3
finding PlotY function, 6-4 to 6-7
functions with new behavior, 12-3
illustration, 3-4
Pop-up Panels, 3-3, 9-6 to 9-7
support for LabWindows for DOS features, 12-2
user interface resource (.uir) files
building, 5-3 to 5-13
adding Command Button, 5-5 to 5-7
adding graph control, 5-7 to 5-9
opening .uir file, 5-4 to 5-5
converting from LabWindows for DOS, 12-3 to 12-4
saving, 5-9
Utility Library
functions with new behavior, 12-3
support for LabWindows for DOS features, 12-2

V

variables
displaying, 4-6 to 4-9
editing, 4-9 to 4-11
Variables window
illustration, 4-7
opening, 4-7
stepping through programs, 4-7 to 4-9
VIC utility, 10-5
View Control Callback command, 7-6
View menu
Find Function Panel command, 3-8
Preview User Interface Header File, 5-9
Recall Function Panel command, 3-10
Source, Interactive Execution, and Standard Input/Output windows
Line command, 2-9
Line Numbers command, 2-8
Next Tag command, 2-9
Toggle Tag command, 2-9
View Variable Value command, Run menu, 4-10
VXI controllers
compatible controllers, 10-4
configuration
configuring LabWindows/CVI for VXI, 10-5 to 10-6
VXI driver software, 10-5
VXI hardware, 10-4 to 10-5
developing applications, 10-5 to 10-6
documentation for VXI controller boards, 10-5
installation, 10-4 to 10-5
LabWindows/CVI VXI development system, 10-4
overview, 10-4

W

Watch window
Add/Edit Watch Expression dialog box, 4-14
displaying variables during program execution, 4-13 to 4-14
purpose and use, 4-13
waveform generation project. *See also* instrument driver programming example.
adding channel control, 9-2 to 9-3
Read Waveform sample function, 8-9 to 8-10
storing waveform on disk, 9-5
user interface (figure), 9-2

Web support from National Instruments, A-1 to A-2
online problem-solving and diagnostic resources, A-1
software-related resources, A-2
Window menu, Project window
Build Errors command, 2-6
overview (table), 2-4
Runtime Errors command, 2-6
windows
managing windows (note), 2-2
program development windows, 2-1 to 2-2
Windows NT/2000/98/95, event function parameter data type
changes, 11-7
source code changes required, 11-7
Worldwide technical support, A-2

Y

Y Array control, 3-5
YGraphPopup function, 3-7 to 3-8