

COMPREHENSIVE SERVICES

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

SELL YOUR SURPLUS

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

 Sell For Cash  Get Credit  Receive a Trade-In Deal

OBSOLETE NI HARDWARE IN STOCK & READY TO SHIP

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



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

 1-800-915-6216

 www.apexwaves.com

 sales@apexwaves.com

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

Request a Quote

 **CLICK HERE**

GPIB-ENET



LabWindows/CVI

Getting Started with LabWindows/CVI

Internet Support

E-mail: support@natinst.com

FTP Site: [ftp.natinst.com](ftp://ftp.natinst.com)

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

Bulletin Board Support

BBS United States: 512 794 5422

BBS United Kingdom: 01635 551422

BBS France: 01 48 65 15 59

Fax-on-Demand Support

512 418 1111

Telephone Support (USA)

Tel: 512 795 8248

Fax: 512 794 5678

International Offices

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

National Instruments Corporate Headquarters

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

Important Information

Warranty

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

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

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

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

Copyright

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

Trademarks

CVI™, natinst.com™, National Instruments™, the National Instruments logo, NI-DAQ™, NI-488.2™, NI-488.2M™, and The Software is the Instrument™ are trademarks of National Instruments Corporation.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

Contents

About This Manual

Organization of This Manual	xiii
Conventions Used in This Manual	xv
The LabWindows/CVI Documentation Set	xvi
Standard Documentation Set	xvi
Optional Manuals	xvii
Related Documentation	xvii
Customer Communication	xvii

PART I

Tutorial—Getting Acquainted with the LabWindows/CVI Development Environment

Chapter 1

Introduction to LabWindows/CVI

Installing LabWindows/CVI	1-1
Note to Sun Users	1-2
How to Proceed	1-2
Learning LabWindows/CVI	1-2
LabWindows/CVI System Overview	1-3
LabWindows/CVI Program Development Overview	1-4
Using C in LabWindows/CVI	1-5
LabWindows/CVI Program Structure	1-5
User Interface	1-6
Program Shell Generation with CodeBuilder	1-6
Program Control	1-6
Data Acquisition	1-6
Data Analysis	1-7

Chapter 2

Loading, Running, and Editing Source Code

A Note about LabWindows/CVI Windows	2-1
Setting Up	2-2
Loading a Project into LabWindows/CVI	2-3
The Project Window	2-5
Running the Project	2-7
Error Messages	2-7

The Standard Input/Output Window.....	2-8
The Source Window.....	2-8
Editing Tools	2-9
Operating Projects with a User Interface.....	2-11

Chapter 3

Interactive Code Generation Tools

Setting Up.....	3-1
The Library Menu	3-2
Accessing the User Interface Library	3-3
Function Panel Fundamentals.....	3-5
Function Panel Controls.....	3-5
Function Panel Help.....	3-5
Drawing a Graph.....	3-6
Inserting Code from a Function Panel	3-7
Analyzing the Data.....	3-9
Output Values on a Function Panel.....	3-10
Recalling a Function Panel	3-11
Finishing the Program.....	3-13
Interactively Executing a Function Panel.....	3-14

Chapter 4

Executing and Debugging Tools

Setting Up.....	4-1
Step Mode Execution	4-3
Breakpoints.....	4-4
Programmatic Breakpoints.....	4-4
Manual Breakpoints	4-6
Displaying and Editing Data	4-6
The Variables Window	4-7
Editing Variables	4-9
The Array Display.....	4-11
Editing Arrays.....	4-12
The String Display	4-13
The Watch Window	4-13

PART II

Tutorial—Building an Application in LabWindows/CVI

Chapter 5

Building a Graphical User Interface

Setting Up	5-2
The User Interface Editor	5-2
Source Code Connection	5-3
CodeBuilder.....	5-3
The Sample Project.....	5-3
Building a User Interface Resource (.uir) File	5-3
Step 1: Opening a .uir File	5-4
Step 2: Adding a Command Button	5-5
Step 3: Adding a Graph Control to the User Interface.....	5-7
Step 4: Saving the .uir File.....	5-9
Step 5: Generating the Program Shell with CodeBuilder	5-11

Chapter 6

Using Function Panels and the Libraries

Setting Up	6-1
Analyzing the Source Code	6-1
The main Function	6-2
The AcquireData Function.....	6-2
The Shutdown Function	6-3
Generating a Random Array of Data.....	6-3
Finding the PlotY Function	6-5
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	7-2
Goals of Session	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.....	8-3
Initializing the Instrument.....	8-4
Configuring the Instrument.....	8-5
Reading Data with an Instrument Driver.....	8-7
Declaring Arrays from Function Panels.....	8-9
Reading the Waveform.....	8-10
Closing the Instrument.....	8-10
Running the Program.....	8-11
Adding the Instrument to Your Project.....	8-13

Chapter 9 Additional Exercises

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

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
The VXI Development System	10-4
Installing and Configuring Your VXI Hardware.....	10-5
Configuring Your VXI Driver Software	10-5
Configuring LabWindows/CVI for VXI	10-6
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
The User Interface Library.....	12-3
The Utility Library.....	12-3
Converting User Interface Resource (.uir) Files.....	12-3
Converting Source Code.....	12-5
Converting Instrument Drivers.....	12-7
Convert the Instrument Driver Function Panels	12-8
Convert the Instrument Driver Header File	12-8
Convert the Instrument Driver Source Code.....	12-8
Converting Loadable Compiled Modules and External Modules	12-9

Appendix A

Customer Communication

Glossary

Figures

Figure 1-1. Relationship of Program Elements	1-5
Figure 2-1. The Project Window	2-1
Figure 2-2. The Source Window	2-2
Figure 2-3. The User Interface Editor Window.....	2-2
Figure 2-4. Open File Dialog Box in Windows 95/NT.....	2-3
Figure 2-5. Open File Dialog Box in Windows 3.1/UNIX.....	2-4
Figure 2-6. sample1.prj in the Project Window.....	2-4
Figure 2-7. Adding a File to a Project	2-6
Figure 2-8. Information in the Project Window	2-7
Figure 2-9. Source Code for the sample1.c File	2-8
Figure 2-10. Split Source Window	2-10
Figure 2-11. sample2.prj Files in the Project Window.....	2-11
Figure 2-12. sample2.uir Panel When Running.....	2-12
Figure 3-1. Library Menu	3-2
Figure 3-2. User Interface Library Function Tree	3-3
Figure 3-3. Y Graph Popup Function Panel	3-4
Figure 3-4. Completed Y Graph Popup Function Panel.....	3-7
Figure 3-5. Source Window with Code Inserted from Function Panel	3-8

Figure 3-6.	Find Function Panel Dialog Box	3-9
Figure 3-7.	Function Panel for Mean	3-10
Figure 3-8.	Completed Function Panel for Mean.....	3-11
Figure 3-9.	Replace/Insert Dialog Box	3-12
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	5-4
Figure 5-3.	Sample User Interface	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.	Completed User Interface.....	5-9
Figure 5-9.	User Interface Header File	5-10
Figure 5-10.	Control Callback Events Dialog Box	5-11
Figure 5-11.	Generate All Code Dialog Box	5-12
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 Subdirectories	1-1
Table 3-1.	Panel Help Display Procedures.....	3-5
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.

Organization of This Manual

Getting Started with LabWindows/CVI is organized as follows:

- Chapter 1, *Introduction to LabWindows/CVI*, contains an overview of the LabWindows/CVI documentation set and the LabWindows/CVI software development system.

Part I: *Tutorial—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.

Part II: *Tutorial—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 have been using throughout this tutorial. Each exercise builds on the code that you developed in the previous exercise. Exercise 1 starts by building on the final sample program that you completed in the Chapter 8 tutorial session. Each exercise gives an outline of the concepts to learn from completing the exercises and some hints for working the exercise.

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.

- The Appendix, *Customer Communication*, contains forms you can use to request help from National Instruments and to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

< >	Angle brackets enclose the name of a key. A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Ctrl-Alt-Delete>.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence File»Page Setup»Options»Substitute Fonts directs you to pull down the File menu, select the Page Setup item, select Options , and finally select the Substitute Fonts option from the last dialog box.
	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
bold	Bold text denotes a parameter, menu item, return value, function panel item, or dialog box button or option.
<i>bold italic</i>	Bold italic text denotes a note, caution, or warning.
<i>italic</i>	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
monospace	Text in this font denotes text or characters that you should literally enter from the keyboard. Sections of code, programming examples, and syntax examples also appear in this font. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, variables, filenames, and extensions, and for statements and comments taken from programs.

paths Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in `drivename\dir1name\dir2name\myfile`.

The LabWindows/CVI Documentation Set

Standard Documentation Set

This manual gives you a hands-on introduction to LabWindows/CVI and shows you how to develop applications with LabWindows/CVI.

The *LabWindows/CVI Instrument Driver Developers Guide* describes how to create instrument drivers for the LabWindows Instrument Library. This manual assumes that you are familiar with the material presented in *Getting Started with LabWindows/CVI* and the *LabWindows/CVI User Manual*.

The *LabWindows/CVI Programmer Reference Manual* contains information to help you develop programs in LabWindows/CVI. This manual assumes that you are familiar with DOS, Windows fundamentals, and with the material presented in *Getting Started with LabWindows/CVI* and the *LabWindows/CVI User Manual*.

The *LabWindows/CVI Standard Libraries Reference Manual* describes the LabWindows/CVI standard libraries—the Analysis Library, the Formatting and I/O Library, the GPIB/GPIB-488.2 Library, the RS-232 Library, and the Utility Library. The *LabWindows/CVI Standard Libraries Reference Manual* assumes that you are familiar with the material presented in *Getting Started with LabWindows/CVI* and the *LabWindows/CVI User Manual*.

The *LabWindows/CVI User Interface Reference Manual* describes how to create custom user interfaces with the LabWindows/CVI User Interface Library. This manual assumes that you are familiar with the material presented in *Getting Started with LabWindows/CVI* and the *LabWindows/CVI User Manual*.

The *LabWindows/CVI User Manual* is a reference manual that describes the features and functionality of LabWindows/CVI.

Optional Manuals

The *LabWindows/CVI Advanced Analysis Library Reference Manual* describes a library of advanced analysis functions. This manual is distributed with the optional LabWindows/CVI Advanced Analysis Library software package.

The *NI-488.2 Function Reference Manual for DOS/Windows*, the *NI-488.2M Function Reference Manual for Win32*, the *NI-488.2M Software Reference Manual*, and the *GPIB32 Online Help* describe a library of functions you can use to program National Instruments GPIB interfaces. This documentation is distributed with National Instruments GPIB products.

The *NI-DAQ User Manual for PC Compatibles* and the *NI-DAQ Function Reference Manual for PC Compatibles* describe a library of functions you can use to program National Instruments DAQ boards. These manuals are distributed with National Instruments DAQ boards.

The *NI-VXI User Manual* and the *NI-VXI Programmer Reference Manual* describe a library of functions you can use to program National Instruments VXI controllers. This manual is distributed with National Instruments VXI controllers for LabWindows/CVI VXI Development System users.

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.

Customer Communication

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

Part I

Tutorial—Getting Acquainted with the LabWindows/CVI Development Environment

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 setup program installs the LabWindows/CVI development environment and a number of additional files on your system. The full installation includes sample programs that illustrate many of the new features in LabWindows/CVI and tutorial programs that you will use throughout this manual. Follow the installation instructions in the *LabWindows/CVI Release Notes* that come with your package to install LabWindows/CVI.

The installer installs LabWindows/CVI and the associated files in subdirectories on your hard disk as shown in Table 1-1.

Table 1-1. LabWindows/CVI Subdirectories

Directory Name	Contents
c:\cvi\bin	LabWindows/CVI Library files
c:\cvi\extlib	Files for using the LabWindows/CVI libraries with external compilers (Windows 95/NT only)
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	SDK library files (Windows 95/NT only)
c:\cvi\toolslib	Additional development tools and libraries
c:\cvi\tutorial	Programs you use in the tutorial sessions throughout this manual.
c:\cvi\vxd	vxd sample code templates (Windows 95/NT only)

Table 1-1. LabWindows/CVI Subdirectories (Continued)

Directory Name	Contents
c:\cvi\misc	Miscellaneous utility files (UNIX only)
c:\cvi\lib	Run-time library files (UNIX only)

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

Note to Sun Users

The figures in the LabWindows/CVI documentation set are from LabWindows/CVI for Windows. However, LabWindows/CVI is a multiplatform application, which means the development environment operates the same on both Windows and the Sun. In some cases, differences exist between Windows and Sun in terms of hot keys or window operation. In cases where these differences affect the tutorial, you will see a note explaining how you should proceed on the Sun.

How to Proceed

The best way to familiarize yourself with LabWindows/CVI is to do the following:

1. Thoroughly read the LabWindows/CVI Release Notes and the `README.CVI` file distributed with LabWindows/CVI.
2. Read the remainder of this chapter for an overall idea of the concepts and capabilities of LabWindows/CVI.
3. Complete the tutorial sessions in Chapters 2–9 as outlined in this manual.
4. Familiarize yourself with the sample programs distributed with LabWindows/CVI.

Learning LabWindows/CVI

LabWindows/CVI is a rich and powerful development environment, featuring libraries to aid you in creating programs for a multitude of data acquisition, test, and measurement applications. Although the LabWindows/CVI documentation is extensive, it is not necessary to completely read all of the LabWindows/CVI manuals to become proficient at using the package.

Beginners should complete this tutorial first. The *LabWindows/CVI User Manual* generally assumes familiarity with *Getting Started with LabWindows/CVI*, and the other manuals assume familiarity with both of these volumes. However, it is still useful to make quick references to other manuals as questions arise, both while learning and while using LabWindows/CVI.

When you begin the tutorial in Chapter 2, *Loading, Running, and Editing Source Code*, of this manual, you will be introduced to the windows, menus, commands, and dialog boxes used in LabWindows/CVI. The *LabWindows/CVI User Manual* contains a chapter devoted to each of the windows in the environment. As the tutorial progresses to such topics as how to use the Variable display window or how to use function panels, remember there are related chapters in the *LabWindows/CVI User Manual*. Scanning these chapters as each topic comes up in *Getting Started with LabWindows/CVI* will help you find the answers to any questions you might have.

As you will see, the tutorial begins with a general introduction to the LabWindows/CVI environment and continues with sections devoted to building a project in LabWindows/CVI. Because each step of the tutorial builds on previous elements, you should follow the outline as given and not skip ahead.

As you work through the tutorial, you may find it helpful to refer to each of the individual library reference manuals as the topics they cover arise. Take time to acquaint yourself with each of the manuals by reading its table of contents. Study the function tree of each manual you will be using in your own development work.

The table of contents of each manual lists the location of helpful information in that manual. The function tree gives you a quick look at how LabWindows/CVI library functions are organized, together with a capsule description of each function.

For a listing of manuals and the LabWindows/CVI topics covered in each, refer to the *About This Manual* section of this manual.

LabWindows/CVI System Overview

LabWindows/CVI is a software development system for C programmers. It contains an interactive environment for developing programs and libraries of functions for creating data acquisition and instrument control applications. LabWindows/CVI contains a comprehensive set of software tools for data acquisition, analysis, and presentation.

You will use the interactive environment for editing, compiling, linking, and debugging ANSI C programs. 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* that you can use to interactively execute the function and generate code for calling the function. Function panels have online help for the function itself and for each control on the function panel.

Programs you write within 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 developing 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.

- For data acquisition, seven libraries exist—the Instrument Library, GPIB/GPIB 488.2 Library, Data Acquisition Library, Easy I/O for DAQ, RS-232 Library, VISA Library, and the VXI Library available in the VXI Development System.
- For data analysis, three libraries exist—the Formatting and I/O Library, Analysis Library, and the optional Advanced Analysis Library.
- For data presentation, LabWindows/CVI offers the User Interface Library.
- For networking and interprocess communication applications, four libraries exist—the Dynamic Data Exchange (DDE) Library (Windows version only), the Transmission Control Protocol (TCP) Library, the X Property Library (UNIX only), and the ActiveX Automation Library (Windows 95/NT only).

In addition, the complete standard ANSI C Library is available within the LabWindows/CVI development environment.

The Instrument Library is a special LabWindows/CVI library. It contains drivers for GPIB, VXI, and RS-232 instruments such as oscilloscopes, multimeters, and function generators. Each driver is distributed in source code so you can modify it if necessary. LabWindows/CVI has all of the development tools for creating your own instrument drivers. Instrument drivers can be created for a single instrument, multiple instruments, or a virtual instrument for which no physical instrument exists. Instrument drivers are created using functions from the other LabWindows/CVI libraries.

The User Interface Library contains tools for controlling graphical user interfaces (GUIs) from your application programs. LabWindows/CVI has a User Interface Editor for creating GUIs and a library of functions for controlling them. With the User Interface Library, you can control panels with 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.

LabWindows/CVI Program Development Overview

While working in LabWindows/CVI, it is important to adhere to the same good programming practices common to all languages and development environments. It is a good idea to do a functional design of your program before you begin writing code. Maintaining good documentation and commenting your code will help you better manage your program development.

Using C in LabWindows/CVI

LabWindows/CVI enhances the C programming language for instrumentation applications. Before you begin working with LabWindows/CVI, be sure that you have a fundamental understanding of C programming. For a description of the ANSI C Standard Library as implemented in LabWindows/CVI, review Chapter 1, *ANSI C Library*, in the *LabWindows/CVI Standard Libraries Reference Manual*.

It is also helpful to examine the sample program listings distributed with LabWindows/CVI. These examples illustrate both the LabWindows/CVI User Interface features and the manner in which you can use the LabWindows/CVI library functions.

LabWindows/CVI Program Structure

After you have made your preliminary exploration of LabWindows/CVI by completing the tutorial in this manual, the next step is to begin your own project. Because LabWindows/CVI is so flexible, you can take almost any approach to building a program. Most programs, however, have a structure that includes some, if not all, of the following elements:

- User Interface
- Program Control
- Data Acquisition
- Data Analysis

Figure 1-1 shows that Program Control gets input from the User Interface, Data Acquisition, and Data Analysis elements, and shows what makes up each element.

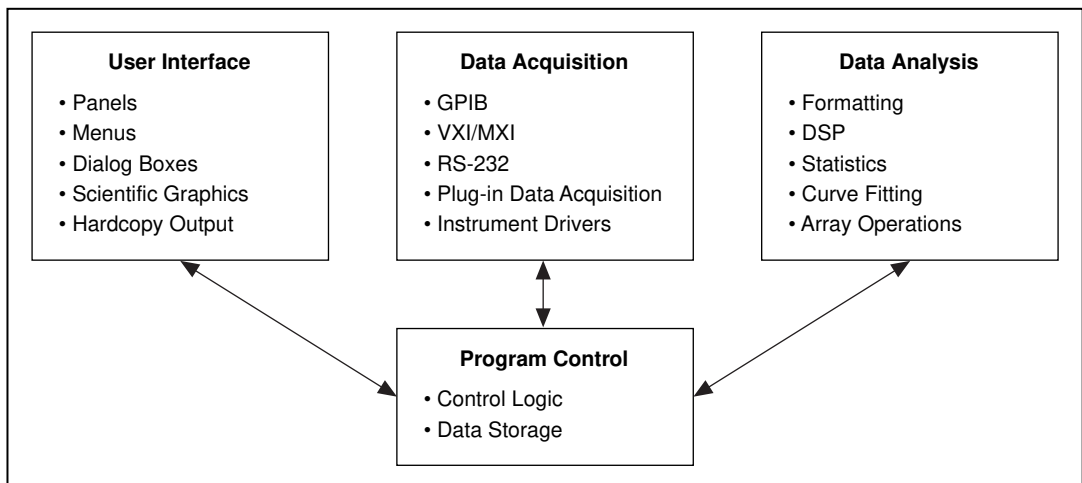


Figure 1-1. Relationship of Program Elements

While Figure 1-1 is only a rough outline, it does provide some orientation for the beginning LabWindows/CVI user.

User Interface

With the LabWindows/CVI User Interface Editor you can build elaborate, interactive panels for your program with a minimum of effort. Designing this graphical user interface (GUI) often confronts you with decisions concerning exactly what you want your program to do. Therefore, the user interface is a natural place to begin your program design.

The user interface provides a mechanism for acquiring input from and displaying output to the user via menus, panels, controls, and dialog boxes.

The elements of the user interface and the functions that allow you to connect your interface to the rest of your program are described in the *User Interface Reference Manual*.

Program Shell Generation with CodeBuilder

After you have designed your GUI in the User Interface Editor, you can automatically generate a program shell based on the components in your GUI using CodeBuilder. CodeBuilder will automatically write code for all of your control callback functions and create 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. The guided tutorial introduces 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, as well as user-defined support functions.

Most of the program control element of a LabWindows/CVI program consists of code you write yourself. Therefore, the code used in the sample programs included in the LabWindows/CVI package is useful in understanding how to create your own program code. You will find that using callback functions in LabWindows/CVI greatly simplifies your task in controlling the flow of your application.

Data Acquisition

Whether you are acquiring data from an instrument or a plug-in data acquisition (DAQ) board, some control of that acquisition will almost certainly be necessary for your program. This portion of your program provides the raw data to be analyzed and presented by other components of your program.

The various LabWindows/CVI libraries provide functions for creating this program element. Functions exist for controlling GPIB, RS-232, and VXI devices, as well as National Instruments DAQ boards. 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 *LabWindows/CVI Standard Libraries Reference Manual*, 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.

Data Acquisition library functions are documented in the *NI-DAQ User Manual for PC Compatibles* and the *NI-DAQ Function Reference Manual for PC Compatibles* distributed with National Instruments DAQ boards. This documentation contains a table describing which functions apply to specific DAQ boards along with LabWindows/CVI-specific information on using those functions.

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, see Chapter 10, *Easy I/O for DAQ Library*, in the *LabWindows/CVI Standard Libraries Reference Manual*.

Information on using the LabWindows/CVI Instrument Library can be found in 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 go into more detail on these topics.

Data Analysis

After you acquire data, it is often necessary to analyze it. Analysis may include formatting, scaling, signal processing, statistics, and curve fitting. The Formatting and I/O Library, the Analysis Library, and the optional Advanced Analysis Library contain functions that perform these operations.

The Formatting and I/O Library and the Analysis Library are described in the *LabWindows/CVI Standard Libraries Reference Manual*.

The optional Advanced Analysis Library is described in the *LabWindows/CVI Advanced Analysis Library Reference Manual*, distributed with the LabWindows/CVI Advanced Analysis Library package.

Loading, Running, and Editing Source Code

In this session of the tutorial, you will load and run various projects in the LabWindows/CVI development environment, and you will learn about the following:

- Some of the windows in LabWindows/CVI
- How to load and operate projects in LabWindows/CVI
- The different types of files that can be used in a LabWindows/CVI project
- Some of the source code editing techniques available in LabWindows/CVI

A Note about LabWindows/CVI Windows

The three main windows you will use in this tutorial are the Project window, the Source window, and the User Interface Editor window. Unlike other windows in the LabWindows/CVI program, the titles of these windows change according to the program you are working on. The file extension is your clue to which window you are in. For example, if you are working on the `sample1` project as you will do next, you will see that the Project window has a `.prj` file extension and is titled `c:\cvi\tutorial\sample1.prj`. When you open a source code file in a project, you get what is called the Source window. This will have a `.c` file extension. The User Interface Editor window has a `.uir` file extension. The top portions of these three windows appear in the following illustrations.

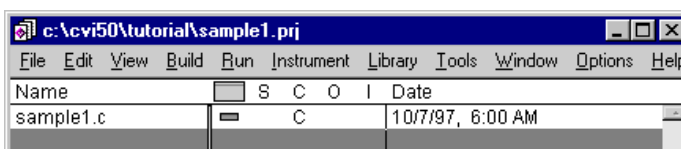


Figure 2-1. The Project Window

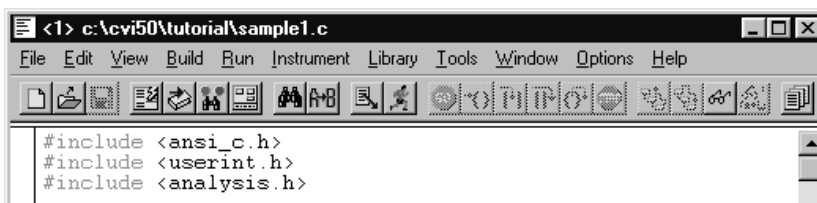


Figure 2-2. The Source Window

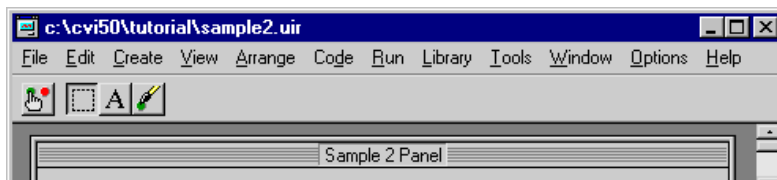


Figure 2-3. The User Interface Editor Window

Setting Up

To begin the LabWindows/CVI tutorial, launch LabWindows/CVI by double-clicking on the LabWindows/CVI icon. After LabWindows/CVI opens, you see an empty Project window.



Note

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

Under UNIX, you can use the title bar window operations available from the window manager installed on your system to manipulate LabWindows/CVI windows.

If other people will be using the Getting Started example files on your computer, be sure to use the Save As option to save your files under different file names.

Loading a Project into LabWindows/CVI

To view some of the editing and execution features of the LabWindows/CVI development, you will load a project into the LabWindows/CVI Project window. Follow these instructions.

1. Select **Open** from the **File** menu. A list appears, showing the different file types that you can create and edit in LabWindows/CVI.
2. Select `Project (*.prj)` to bring up the Open File dialog box shown in Figure 2-4 and Figure 2-5. You may have to click on the `CVI` directory, then click on the `tutorial` directory to bring up the project files.

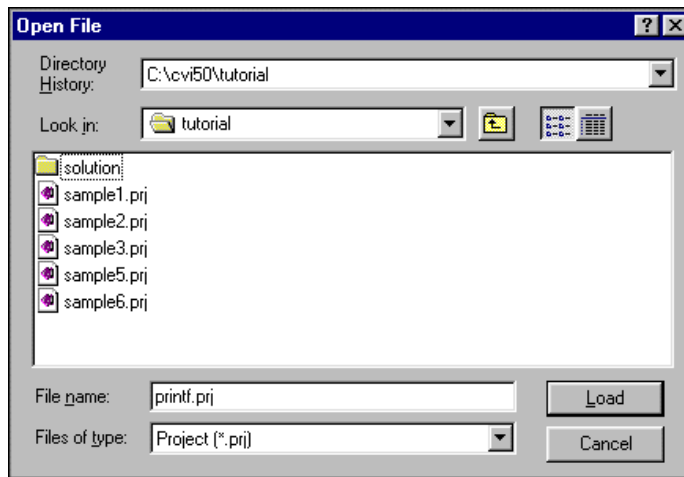


Figure 2-4. Open File Dialog Box in Windows 95/NT

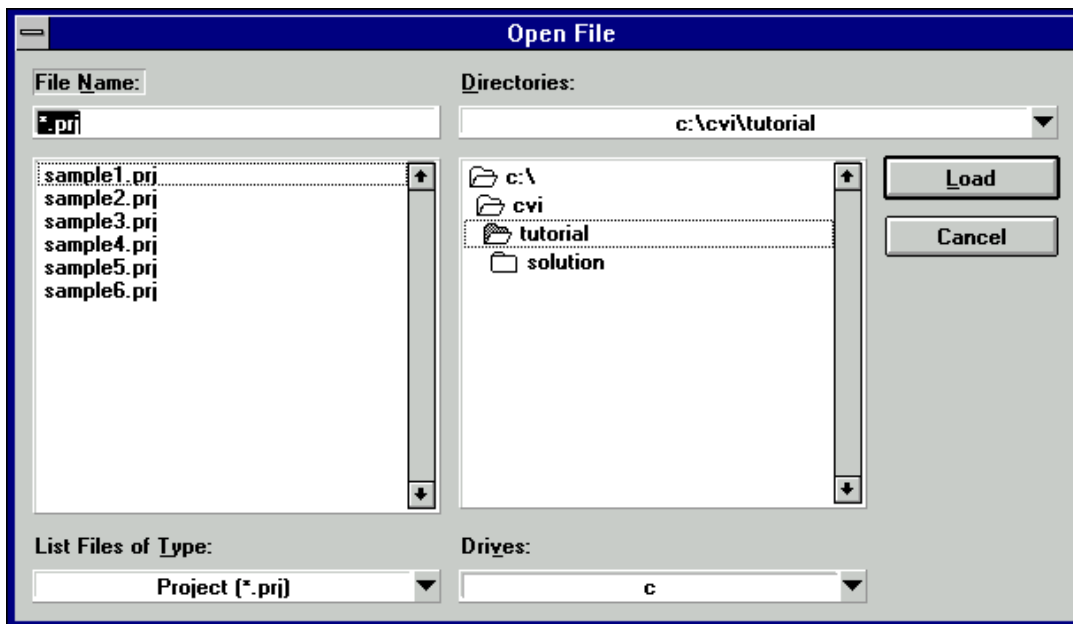


Figure 2-5. Open File Dialog Box in Windows 3.1/UNIX

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

After you load the `sample1.prj` project, the Project window should appear as shown in Figure 2-6.



Note

If you are not the first person to use this tutorial on your computer, it might be necessary to reinstall LabWindows/CVI in order to get the unmodified versions of the tutorial code.

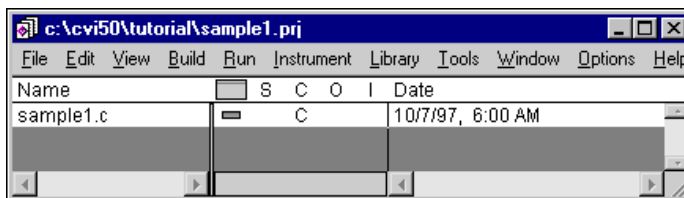


Figure 2-6. `sample1.prj` in the Project Window

Now let us look at some of the various windows available in LabWindows/CVI: the Project window, the Standard Input/Output window, and the Source window.

The Project Window

The Project window in LabWindows/CVI lists all of the files that make up a particular project or program.

- The **File** menu contains selections for loading, saving, and creating new files in LabWindows/CVI. You can open any type of file (project, source, header, or user interface) from the **File** menu in the Project window.
- The **Edit** menu contains selections for adding or removing files from the project list.
- The **Build** menu contains selections for using the compiler and linker in LabWindows/CVI.
- The **Run** menu contains selections for running a project.
- The **Window** menu lets you go to other windows in LabWindows/CVI quickly, such as the Source window, User Interface Editor window, and Standard Input/Output window.
- The **Options** menu contains selections for configuring various aspects of the LabWindows/CVI programming environment. You will be introduced to many of these windows, utilities, and editors throughout this tutorial. For more information on the Project window menus, see Chapter 3, *Project Window*, of the *LabWindows/CVI User Manual*.

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

Projects that you build with LabWindows/CVI can consist of multiple files of many different types. LabWindows/CVI is compatible with C source files, object modules, Dynamic-Link-Libraries (DLLs), C libraries, user interface files, and instrument drivers. When you develop more advanced projects with LabWindows/CVI, you simply select **Add File to Project** from the **Edit** menu and select the type of file that you would like to add to your project, as shown in Figure 2-7.

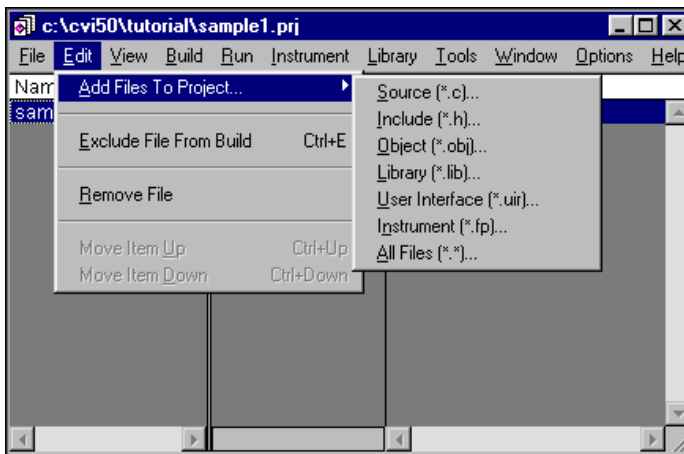


Figure 2-7. Adding a File to a Project

The project you just opened, `sample1.prj`, is a very simple project, consisting only of a single C source file. The Project window displays status information for the files listed in the project list. Figure 2-8 shows how this information is displayed in the Project window.

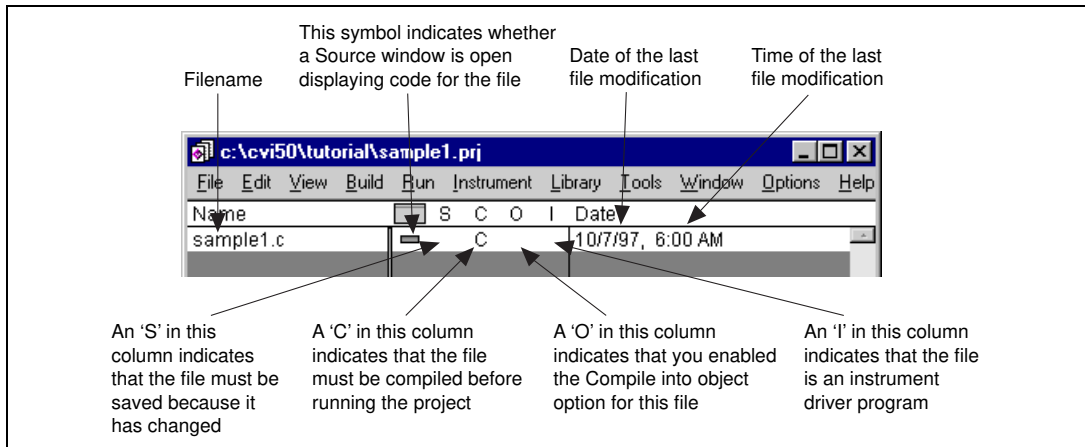


Figure 2-8. Information in the Project Window

Running the Project

To run the `sample1` project, select **Run Project** from the **Run** menu. LabWindows/CVI automatically compiles any source files in the project list, links the project with the libraries used, and executes the compiled code. Notice that the `C` indicator disappears from the project list after the source compiles. When the project begins running, the word **Running** appears in the upper left-hand corner of the Project window.

The `sample1` project is a very 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 will appear at the bottom of your screen. The type of error you have will be highlighted, and the line number of the error will be to the left of the highlighted error type. Correct your error and rerun your program.

To remove the error message window from the screen, double-click in the Close box in the upper left-hand corner of the window. Select **Build Errors** or **Runtime Errors** from the **Window** menu to make the **Error Window** reappear.

The Standard Input/Output Window

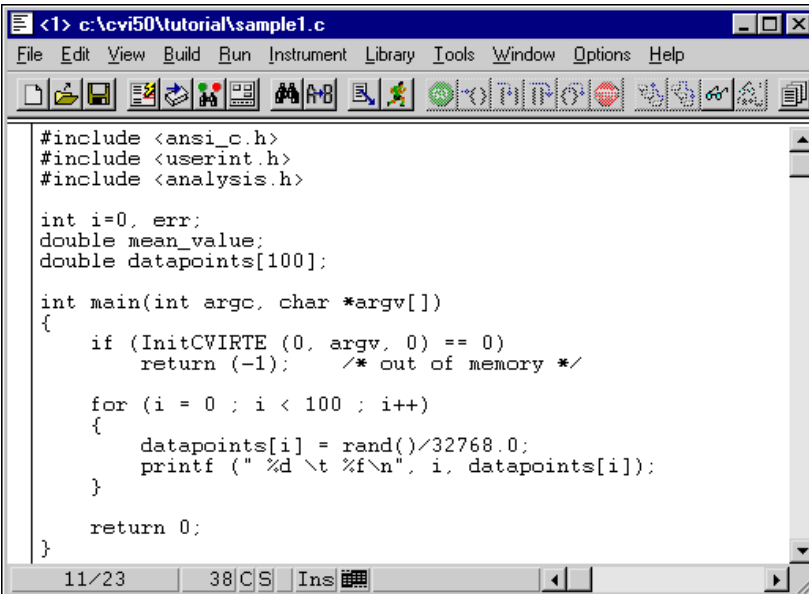
The Standard Input/Output window is where simple, text-based information is displayed to or received from the user during program execution. If you want to use the ANSI C `stdio` library when developing your C programs in LabWindows/CVI, the results of the `printf` and `scanf` functions appear in the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

 **Note** *To display the standard I/O window whenever text is written to it, set the **Bring Standard Input/Output to front whenever modified** option in the **Project window's Option»Environment dialog box**.*

The Source Window

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

1. Close the Standard Input/Output window by selecting **Hide** from the **File** menu, or by pressing `<Ctrl-W>`.
2. Double-click on the `sample1.c` filename in the project list to display the source code in a source window.
3. The source code should appear as shown in Figure 2-9. As you can see, the source code for the `sample1` file contains standard ANSI C compatible code.



```

<1> c:\cvi50\tutorial\sample1.c
File Edit View Build Run Instrument Library Tools Window Options Help
#include <ansi_c.h>
#include <userint.h>
#include <analysis.h>

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

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

    for (i = 0 ; i < 100 ; i++)
    {
        datapoints[i] = rand()/32768.0;
        printf (" %d\t %f\n", i, datapoints[i]);
    }

    return 0;
}
11/23 38|C|S Ins

```

Figure 2-9. 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 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 4, *Source, Interactive Execution, and Standard Input/Output 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 will learn more about code generation tools in later sessions of this tutorial.

The Source Window, Function Panel Windows, and Variable 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 item the icon represents. You can customize the icons in your toolbar from the **Options** menu by selecting **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. If you select the **Edit** menu, you will notice many standard Windows editing features, such as **Cut**, **Copy**, **Paste**, **Find**, and **Replace**. In addition, the arrow positioning keys, <Page Up>, <Page Down>, <Home>, and <End> keys, operate in the Source window in a fashion similar to a word processor.

1. If you are viewing a large file, you may need to refer to particular line numbers. Select **Line Numbers** from the **View** menu. 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 **Open Quoted Text** from the **File** menu, or by pressing <Ctrl-U>.

Place the cursor on the `ansi_c.h` filename on line 1 of the `sample1.c` file 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 **Close** from the **File** menu. The `sample1.c` file should be 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 a top and bottom half 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 should see a duplicate copy of the source code in each subwindow as shown in Figure 2-10.

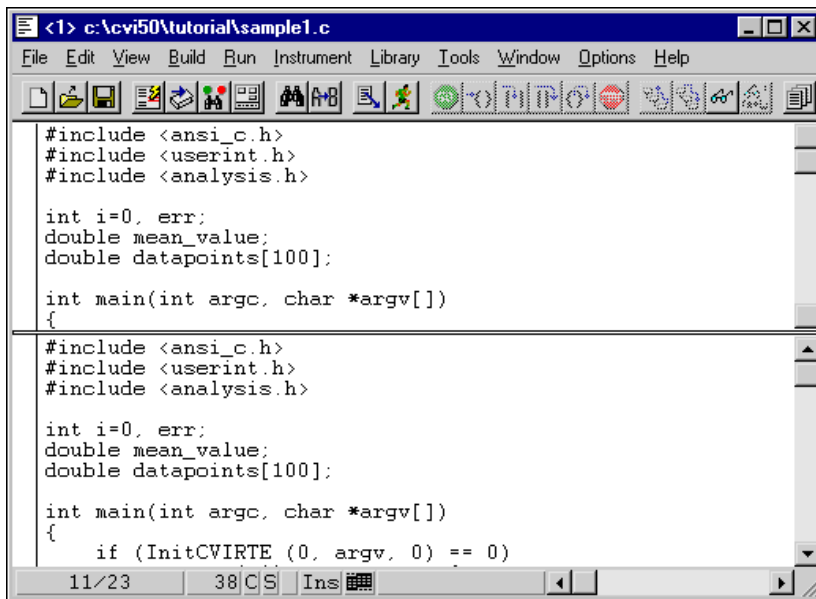


Figure 2-10. Split Source Window

4. Notice how each half of the window can be scrolled independently to view 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 quickly reverse any mistakes you may make. The default configuration of LabWindows/CVI allows up to 100 **Undo** operations. Select **Undo** from the **Edit** menu. The text you entered in step 4 on line 5 of the source code should disappear.

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 **Line** from the **View** menu and enter the line number. Setting tags on particular lines is another method to highlight lines of code to which you can quickly jump.

Place the cursor on line 3. Select **Toggle Tag** from the **View** menu. 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. By selecting **Next Tag** from the **View** menu, your cursor will automatically jump to the next tagged line in your source code. You can also jump between tags by pressing the <F2> key.

Close `sample1.c` before moving on to the next session.

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 graphical user interface (GUI) to control the program flow and display the results. In the Chapter 5 tutorial session you will learn how to build a GUI. The purpose of this session is to briefly show you how a GUI looks and works. Take the following steps to load a new sample program.

1. Select **Open** from the **File** menu and choose `Project (*.prj)` as the file type. Choose `sample2.prj` from the Open File dialog box. You might be prompted to save changes from `sample1.prj`. Click on **Discard** to continue. The Project window should appear as shown in Figure 2-11.

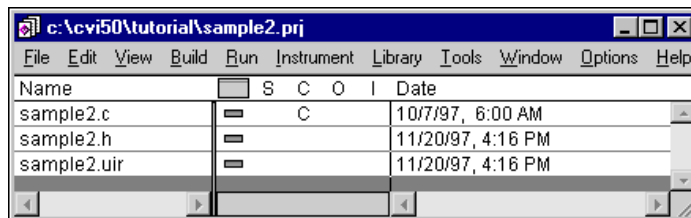


Figure 2-11. sample2.prj Files in the Project Window

2. Run the project by selecting **Run Project** from the **Run** menu, or by entering <Shift+F5>. The GUI shown in Figure 2-12 should appear after the program compiles and runs.

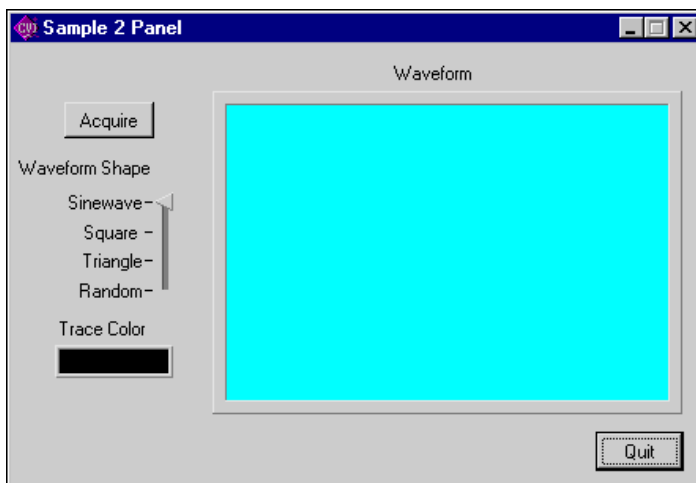


Figure 2-12. sample2.uir Panel When Running

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

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

This concludes the first session of the tutorial. In the next session, Chapter 3, *Interactive Code Generation Tools*, you will learn how to use interactive code generation tools.

Interactive Code Generation Tools

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

Setting Up

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

1. Close all windows except the Project window.
2. Select **Open** from the **File** menu and choose `Project (*.prj)` as the file type.
3. Open the `sample1` project file in one of these three ways: type `sample1.prj` in the File Name input and press <Enter> or click on the **Load** button; double-click on the `sample1.prj` file listed in the dialog box; or select `sample1.prj` and press <Enter> or click on the **Load** button.

If you have not saved the previous contents of the windows, you will be prompted to do so. If you want to save the previous contents, click on the **Save** command button and enter a filename into the File Name input box. If you do not want to save the previous contents, click on the **Discard** command button.

Now let us look at the **Library** menu.

The 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. By breaking down each library into a hierarchy, you can find your way around the library functions more easily.



Note *Depending on which package you have, the submenus under the Library menu will include either the Analysis library or the Advanced Analysis Library.*

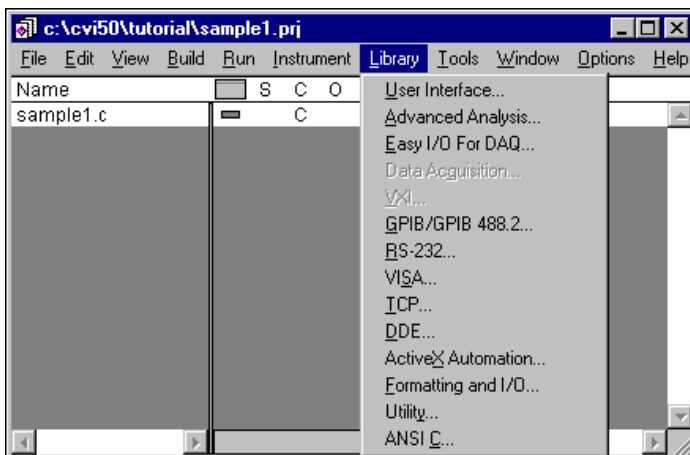


Figure 3-1. Library Menu

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

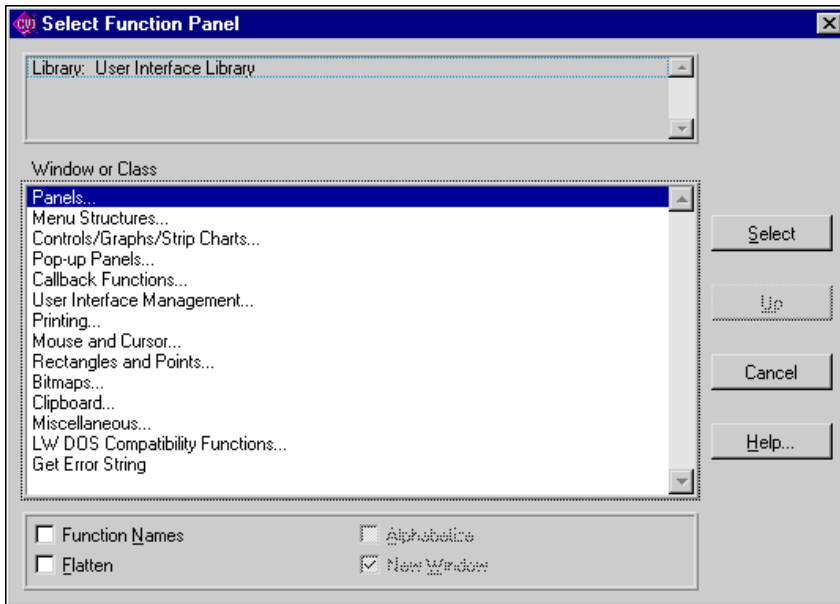


Figure 3-2. User Interface Library Function Tree

Accessing the User Interface Library

You will now use the User Interface Library to display a graph of the random numbers you generated in the `sample1` project. Follow these steps. Close the User Interface Library window if you opened it in the previous section.

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 Project** from the **Run** menu. One hundred numbers should be output to the Standard Input/Output window (0–99).

3. After program execution ends, close the Standard Input/Output window.
4. Position the input cursor before the line `return 0;` in the source file (line 19). This line is located before the closing right brace (`}`) for the `for` loop.
5. Select **User Interface...** from the **Library** menu. A dialog box appears.
6. Press `<P>` once to select **Pop-up Panels**, or use the down arrow key, then press `<Enter>`. Another dialog box appears.
7. Press `<Y>` to select the `YGraphPopup` item, or use the arrow key, then press `<Enter>`.

After selecting `YGraphPopup`, a new screen appears as shown in Figure 3-3.

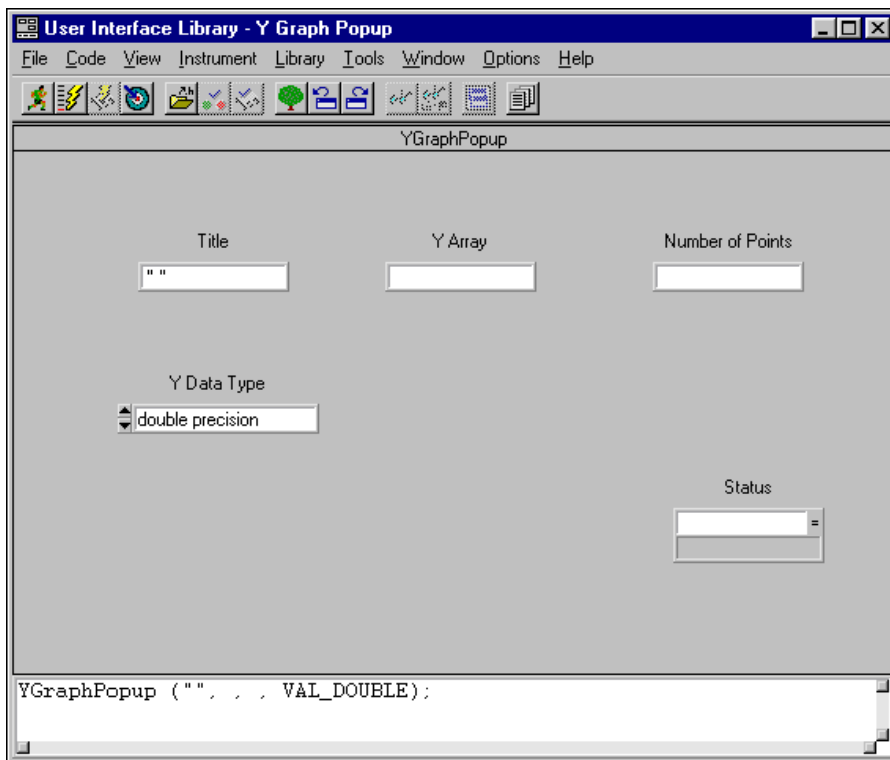


Figure 3-3. Y Graph Popup Function Panel

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.

1. 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.
2. With function panels, you can automatically declare variables in memory to be used as function parameters.
3. 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.
4. Function panels generate code automatically, so that the function call syntax is automatically inserted into your program source code.

Function Panel Controls

The items on the function panel are called controls that you manipulate to specify parameters. There are eight types of controls, and these controls are explained as you encounter them in the examples that follow.

The highlight should be on 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 highlight from one control to another. With a mouse, click on the label of the desired control to move the highlight.

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. The methods for accessing this help information are listed in the following table.

Table 3-1. Panel Help Display Procedures

Type of Help	How to View Help
Function Help	Select Function from the Help menu <i>or</i> Right click anywhere on the Function panel.
Control Help	Highlight the control, then select Control from the Help menu <i>or</i> Click the right mouse button on the desired control.

Select **Function** from the **Help** menu to view information pertaining to the `YGraphPopup` function panel. Press `<Enter>`, or click on the **Done** command button, to remove the dialog box.

Select the Y Array control. Press `<F1>`, or click the secondary mouse button on the Y Array control, to view the control-specific help information. After reading the help information, remove the dialog box.

Drawing a Graph

You will now use the function panel to create a line of code that will graph the array of random numbers generated in the sample program. Move the highlight to the Title control. Then perform the following steps. Remember, you can use the `<Tab>` key to move from one control to another, or you can click on a control with your mouse to highlight it.

1. Enter "Random Data" in the Title control. Be sure to include the quotation marks.
2. Press `<Tab>` to move the highlight control to the Y Array control.
3. Enter the array name `datapoints` in the Y Array control.
4. Press `<Tab>` to move the highlight to the Number of Points control.
5. Enter the number `50` in the control.
6. Press `<Tab>` to move the highlight to the Y Data Type control. As you can see by the up and down arrows, this is a ring control.
7. Use the up and down arrow keys on your keyboard to move through the choices until you find `double precision`. To use the mouse, click on the ring control and select `double precision` from the pull-down menu, or click on the arrow key to find `double precision`.
8. Press `<Tab>` to move the highlight to the Status control or click on the Status control.
9. Enter the variable name `err` in the control.

The function panel appears as shown in Figure 3-4.

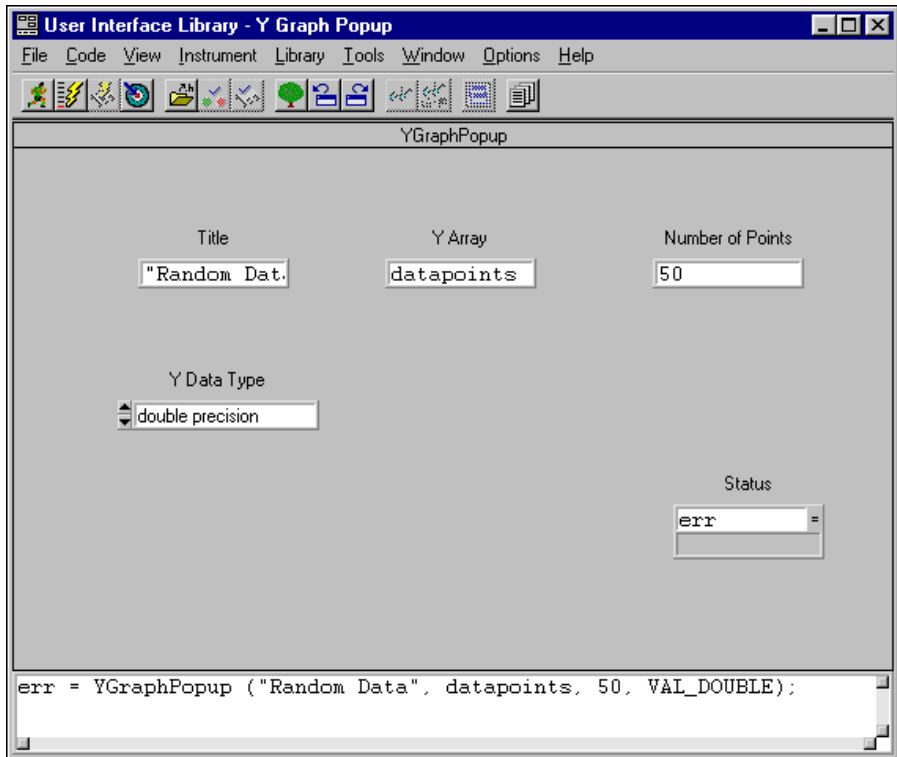


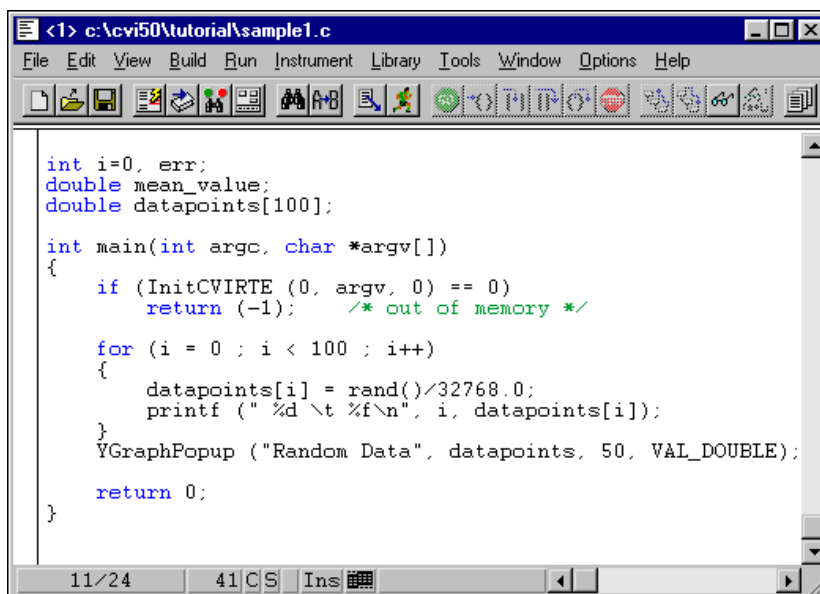
Figure 3-4. Completed Y Graph Popup Function Panel

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 when you manipulate the controls on the function panel. You can copy these lines of code directly to your application program by completing the following steps.

1. Select **Set Target File** from the **Code** menu to bring up a dialog box. Select `sample1.c` from the list of windows in the dialog box.
2. Select **Insert Function Call** from the **Code** menu. When you make this selection, LabWindows/CVI automatically 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. Select **Close** from the **File** menu to remove the YGraphPopup Function Panel window.

- The `sample1.c` source code should have the `YGraphPopup` function added at the bottom of the program, as shown in Figure 3-5. Notice the line of code inserted from the function panel.



```

<1> c:\cvi50\tutorial\sample1.c
File Edit View Build Run Instrument Library Tools Window Options Help

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

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

    for (i = 0 ; i < 100 ; i++)
    {
        datapoints[i] = rand()/32768.0;
        printf (" %d\t%f\n", i, datapoints[i]);
    }
    YGraphPopup ("Random Data", datapoints, 50, VAL_DOUBLE);

    return 0;
}
11/24 41|CS| Ins

```

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

- Execute the program by selecting **Run Project** from **Run** menu. Click on **Yes** if you see a dialog box asking 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. Press <Enter> or click on **OK** to remove the graph and return to the Source window.
- Close the Standard Input/Output window.

Analyzing the Data

Now you will use a function from the LabWindows Analysis Library to calculate the mean of the values in the array. Before continuing, position the input cursor in the Program 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 **Find Function Panel** from the **View** menu. A dialog box appears as shown in Figure 3-6.

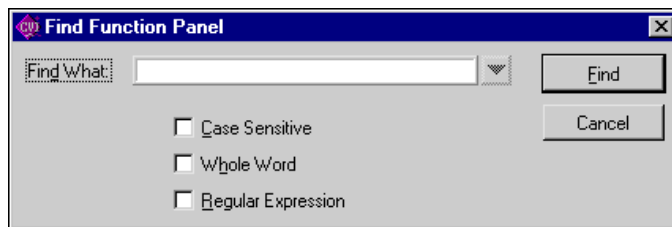


Figure 3-6. Find Function Panel Dialog Box

2. Type `Mean` into the Function Name input box.
3. Press <Enter>, or click on the **Find** command button to search for the `Mean` function panel.

When LabWindows/CVI finds the `Mean` function, its function panel appears, as shown in Figure 3-7.

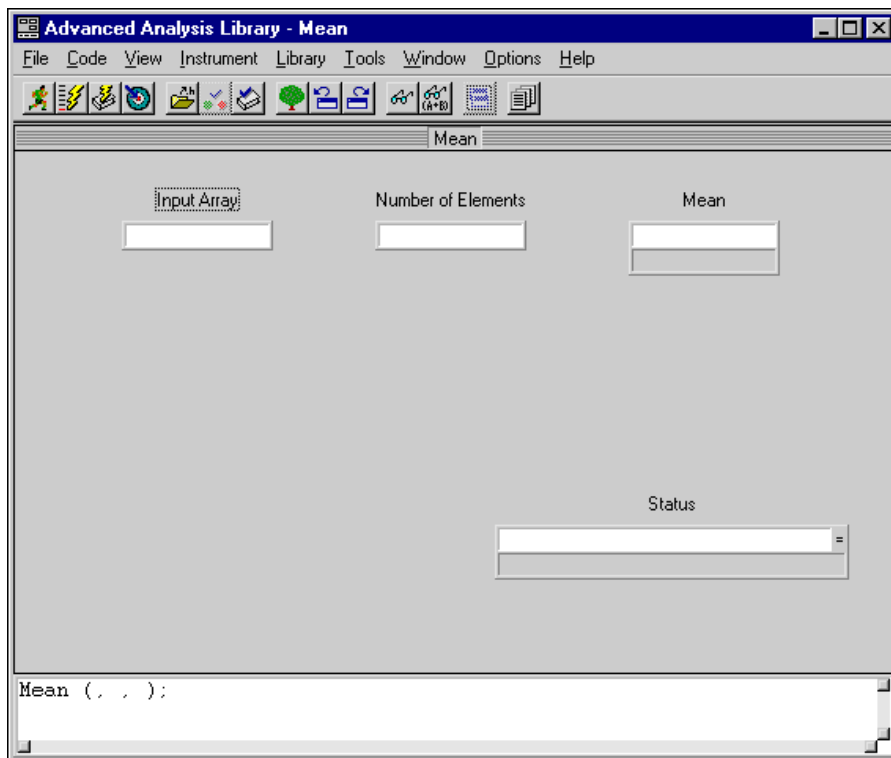


Figure 3-7. Function Panel for Mean

The **Input Array** label should be highlighted. If it is not, press <Tab> to move the highlight to this label. Or with a mouse, click on the label to highlight it. Make the following entries on the function panel.

1. Enter the array name `datapoints` in the Input Array control.
2. Press <Tab> to move the highlight to the Number of Elements control.
3. Enter the number `100`.

Leave the remaining control empty and go on 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. Do this if you intend to generate code for your program. For this example, enter a variable name as follows.

1. Select the Mean control by tabbing or using the mouse.
2. Enter `&mean_value` in the control. The function appears as shown in Figure 3-8.

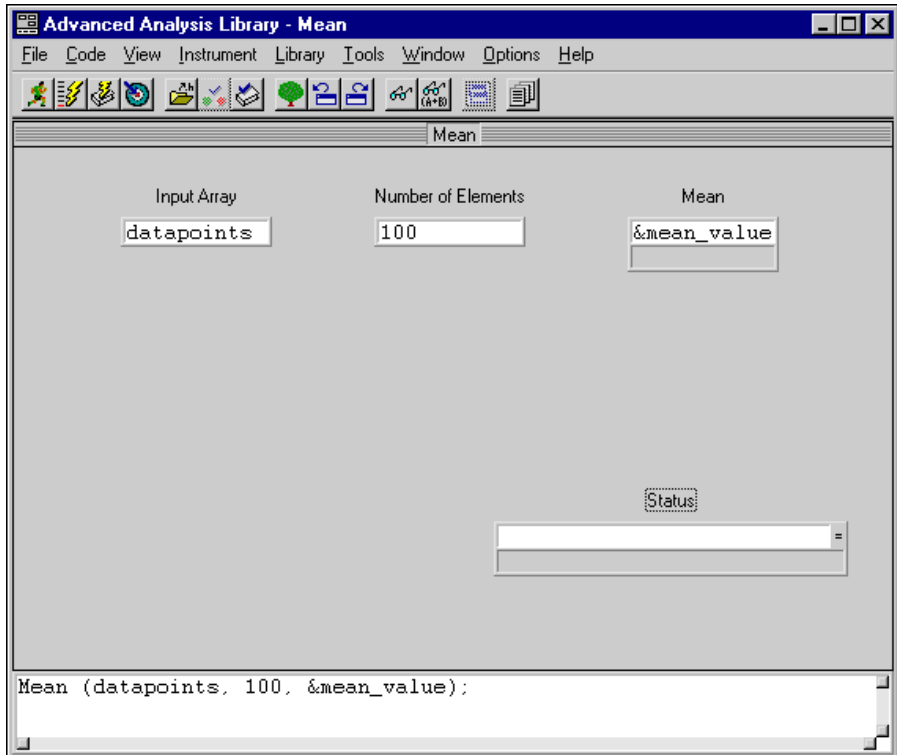


Figure 3-8. Completed Function Panel for Mean

3. Insert the line of code for calling the `Mean` function into the `sample1.c` source code by selecting **Insert Function Call** from the **Code** menu, or by pressing `<Ctrl-I>`.
4. Select **Close** from the **File** menu to remove the Mean function panel. You will see the code where you left your cursor in the Source window.

Recalling a Function Panel

Notice that the 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 Program 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. Position the input cursor in the Program window on the following line.

```
err = YGraphPopup ("Random Data", datapoints, 50, VAL_DOUBLE);
```
2. Select **Recall Function Panel** from the **View** menu. The YGraphPopup function panel appears on the screen. Notice that the controls are automatically adjusted to reflect the state of the line of code from the Program window.
3. Select the Number of Points control.
4. Type 100 in the control.
5. Copy the new code to your program by selecting **Insert Function Call** from the **Code** menu. The Replace/Insert dialog box appears as shown in Figure 3-9.

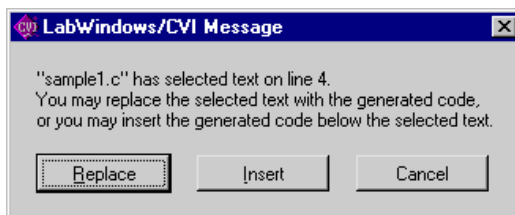


Figure 3-9. Replace/Insert Dialog Box

6. Press <Enter>, or click on the **Replace** command button, to replace the old line of code in the Program window with the newly generated line of code.
7. Select **Close** from the **File** menu to remove the Function Panel window and return to the Program window.

Notice that the call to YGraphPopup will now graph 100 elements of the array datapoints.

Finishing the Program

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

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

The completed program listing is shown in the following illustration.

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

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

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

    for (i = 0 ; i < 100 ; i++)
    {
        datapoints[i] = rand()/32768.0;
        printf ("%d \t %f\n", i, datapoints[i]);
    }

    YGraphPopup ("random data", datapoints, 100, VAL_DOUBLE);
    err = Mean (datapoints, 100, &mean_value);
    printf("Mean = \t %f \n", mean_value);

    return 0;
}
```

Execute the program by selecting **Run Project** from the **Run** menu. If a dialog box comes up asking you to save changes before running your program, click on **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 (line 99).

Close your Source window before going on to the next tutorial session.

Interactively Executing a Function Panel

In this tutorial session, you have learned how to use function panels to interactively build function calls into your program. You have also learned that function panels can teach you how functions operate through the online help. Perhaps the most powerful feature of function panels is that of executing functions interactively without making the function call part of a program. In Chapter 6, *Using Function Panels and the Libraries*, and Chapter 7, *Adding Analysis to Your Program*, of this tutorial, you will learn how to use function panels to declare variables for use in your program and how to run functions interactively.

This concludes the second tutorial session. In the next session you will learn how to use the executing and debugging tools available in LabWindows/CVI.

Executing and Debugging Tools

In this session, you will get acquainted 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.

Setting Up

1. Open the project in the `Tutorial` directory titled `sample3.prj`.
2. Close all windows except the Project window displaying the files for `sample3.prj`.
If you have not saved the previous contents of these windows, you will be prompted to do so. If you want to save the previous window contents, click on the **Save** command button and enter the desired filename into the Filename input box. If you do not want to save the previous window contents, click on the **Discard** command button.
3. Double-click on the `sample3.c` filename in the project list to open a source window.

The program `sample3.c` appears in the Source window as shown in Figure 4-1.

```

<1> c:\cvi50\tutorial\sample3.c
File Edit View Build Run Instrument Library Tools Window Options Help

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

void get_and_print_random (int, double *);

int main (int argc, char *argv[])
{
    double my_array[100];
    int i;

    if (InitCVRTE (0, argv, 0) == 0)
        return (-1); /* out of memory */

    Cls ();
    for (i = 0; i < 100; i++)
    {
        get_and_print_random (i, &my_array[i]);
    }

    return 0;
}

void get_and_print_random (int index, double *random_val)
{
    *random_val = (double)rand () / 32767.0;
}
28/29 54|C|S| Ins

```

Figure 4-1. `sample3.c` Program

The program `sample3.c` performs the same random number function as the program `sample1.c` that you ran in the first session of this tutorial in Chapter 2, [Loading, Running, and Editing Source Code](#).

The difference between the programs is that in `sample3.c`, the steps of assigning a random number and printing out the values are placed in a function. This arrangement of program instructions is useful in illustrating 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 **Break at First Statement** from the **Run** menu to stop execution at the first line in the source code. When **Break at First Statement** mode is activated, a checkmark (✓) appears next to the **Break at First Statement** item in the **Run** menu.



2. Select **Run Project** from the **Run** menu, click on the **Run Project** toolbar icon, or press <Shift+F5> to begin execution of the program. After the program is compiled, the `main` function line in the program appears highlighted in the Source window, indicating that program execution is currently suspended.

You can halt execution of the program during step mode by selecting **Terminate Execution** from the **Run** menu, clicking on the **Terminate Execution** toolbar icon, or using quick keys: <Ctrl-Alt-SysRq> under Windows 3.1 and <Ctrl-F12> under Windows 95/NT and UNIX. Do not halt execution now.

3. To execute the highlighted line, select the **Step Into** command from the **Run** menu or click on the **Step Into** toolbar icon. Notice that the shortcut key combination for the **Step Into** command is <F8>. This is a very useful shortcut that eliminates having to access the **Run** menu to execute each step.

Other stepping operations include **Step Over**, <F10>, and **Continue**, <F5>. **Step Over** executes a function call without single-stepping through the function code itself, as opposed to the **Step Into** selection that single-steps through the code of the function call being executed. **Continue** causes the program to continue operation until it completes or reaches a breakpoint.

4. Any time that you are working with user defined variables or functions in LabWindows/CVI, such as the `get_and_print_data` function in `sample3.c`, you can immediately locate the definition of the function by using the **Go To Definition** toolbar icon. To immediately find the definition of the `get_and_print_data` function, highlight the call to the function on line 17 of `sample3.c` by double-clicking on it, then click on the **Go To Definition** toolbar icon or selecting **Go To Definition** from the **Edit** menu.

Notice that LabWindows/CVI immediately finds the definition of the function. This holds true even if the function resides in a different source file, as long as the source file is listed in the Project window. You can also find variable declarations using this command. Try finding the declaration statements for the variables in `sample3.c` using the **Go To Definition** toolbar icon.

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 several random values have been created.

Breakpoints

Breakpoints are another run-time tool for debugging programs in the LabWindows/CVI environment. A breakpoint is a location in a program at which execution of the program is suspended. Four ways exist to invoke a breakpoint in LabWindows/CVI:

- Programmatic Breakpoint—Inserting a Breakpoint icon in the source code
- Manual Breakpoint—Pressing <Ctrl-Alt-SysRq> under Windows 3.1 during program execution, or Pressing <Ctrl-F12> under Windows 95/NT and UNIX
- Breakpoint on Error—Pause when a LabWindows/CVI library function returns an error
- Conditional Breakpoint—Pause when a user-specified condition becomes true

Programmatic Breakpoints and Manual Breakpoints will be discussed in the following sections. For additional information on Conditional Breakpoints and Breakpoint on Error, see the *Run Menu* section of Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, in the *LabWindows/CVI User Manual*.

Programmatic Breakpoints

To invoke a breakpoint at a specific location in a program, left-click with your mouse in the left-hand column of the Source window on the line you want to break on. In this example, you will insert a breakpoint inside the `for` loop so that the program halts after returning from the function call. Insert a breakpoint in the sample program as follows.

1. Halt program execution, if you have not already done so, by selecting **Terminate Execution** from the **Run** menu or clicking on the **Terminate Execution** toolbar icon.
2. Disable **Break at First Statement** from the **Run** menu so that the checkmark next to it disappears.
3. Left-click with the mouse in the left-hand column of the Source window to the left of the line containing the following statement.

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

4. A red diamond, representing a breakpoint, should appear beside that line as shown in Figure 4-2.

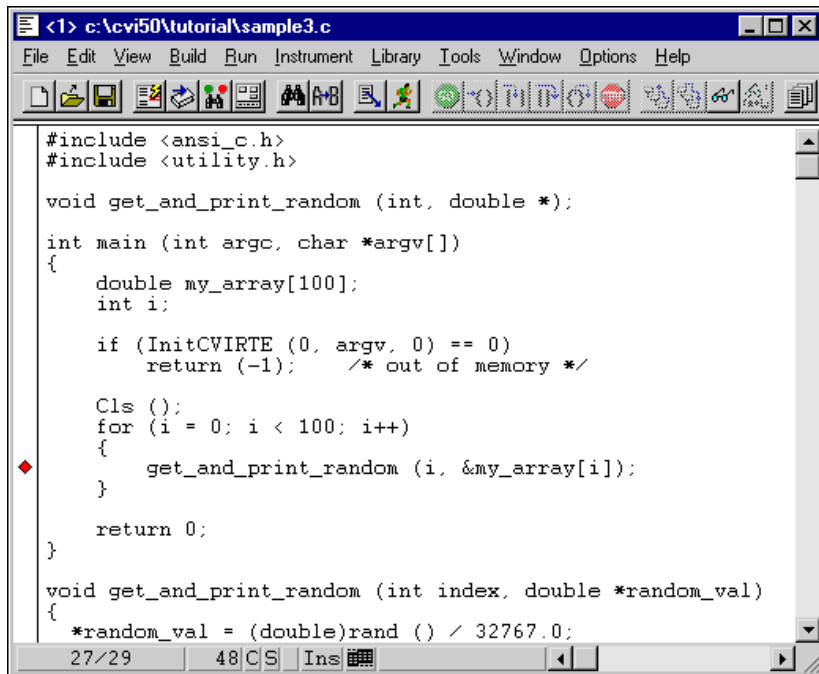


Figure 4-2. Breakpoint beside a Line of Code

To illustrate how the breakpoint works, begin execution of the program by selecting **Run Project** from the **Run** menu. When the breakpoint is encountered during execution, the program execution is suspended and the line with the breakpoint is highlighted with a red box.

Press <F5> to continue execution. Program execution resumes until it encounters a breakpoint or it completes. You can also single-step through the code at that point by selecting **Step Over** or **Step Into** from the **Run** menu.

To halt the program at a breakpoint, press <Ctrl-Alt-SysRq> under Windows 3.1 or <Ctrl-F12> under Windows 95/NT and UNIX, or select **Terminate Execution** from the **Run** menu. Halt the program now using one of these methods.

To remove the breakpoint from the program, left-click on the red diamond so that it disappears.

Manual Breakpoints

You can also enter a breakpoint after program execution has begun by pressing <Ctrl-Alt-SysRq> under Windows 3.1 or <Ctrl-F12> under Windows 95/NT and UNIX. Select **Run Project** from the **Run** menu to begin program execution. When the program begins running, press <Ctrl-Alt-SysRq>, or <Ctrl-F12> under Windows 95/NT and UNIX. The program enters breakpoint mode just as it did when the breakpoint symbol was encountered, but with one difference—instead of the line with the breakpoint symbol being highlighted, the next executable statement in the program appears highlighted. All the execution options at this point are the same as the options available when a call to the `breakpoint` function is encountered in the program. Press <Ctrl-Alt-SysRq> (<Ctrl-F12> under Windows 95/NT and UNIX), or select **Terminate Execution** from the **Program** menu to halt the program.

Displaying and Editing Data

Step mode execution and breakpointing are useful tools for high-level testing. In many cases, however, you need to look deeper than your source code to test your programs. The LabWindows/CVI interactive environment has special displays for viewing and editing data. These displays are the Variables window, the Array Display, and the String Display. In addition, LabWindows/CVI has a utility called the Watch window that lets you view variable and expression values during program execution. The following sections describe each display.

The Variables Window

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

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

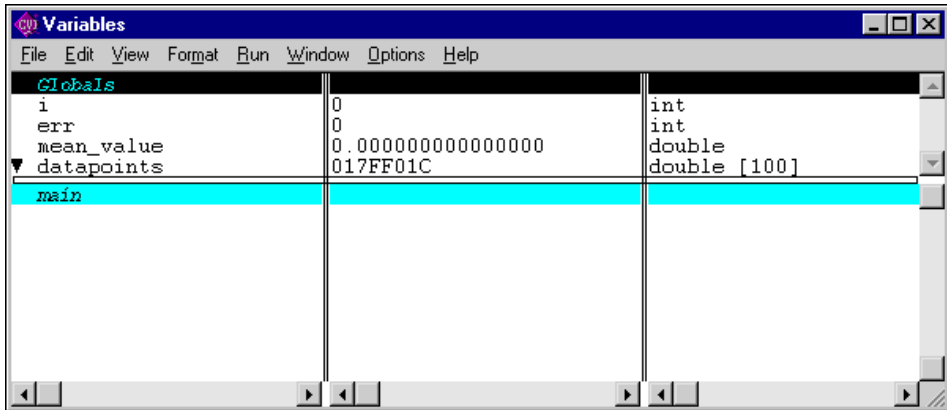


Figure 4-3. Variables Window

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 the preceding illustration shows that there are a number of global or local variables currently declared in two different source files.

You can view the Variables window at any time to inspect variable values. This is especially useful when you are stepping through a program during breakpointed execution. 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 the **Break at First Statement** command from the **Run** menu. A checkmark (✓) should appear before the **Break at First Statement** command in the **Run** menu.
2. Select **Run Project** from the **Run** menu, or press <Shift+F5> to begin running the program. When the program begins execution, the `main` function in the program is highlighted.

3. Select **Variables** from the **Window** menu. The Variables window appears as shown in Figure 4-4.

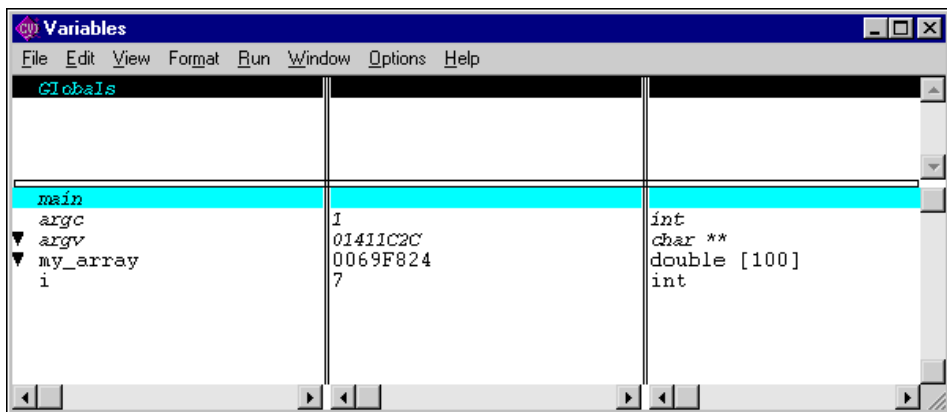


Figure 4-4. Variables Window during Execution of main

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



Note *Your values may differ from the values shown in the preceding illustration.*

4. Select **Hide** from the **File** menu, or double-click in the **Close** box in the upper left-hand corner of the window, to return to the Source window.
5. Select **Step Into** from the **Run** menu or press <F8> repeatedly until the highlight is on the line `*random_val = (double)rand () / 32767.0;`, which is the first statement in the function `get_and_print_random`.

- Select **Variables** from the **Window** menu. The Variables window now appears as shown in Figure 4-5.

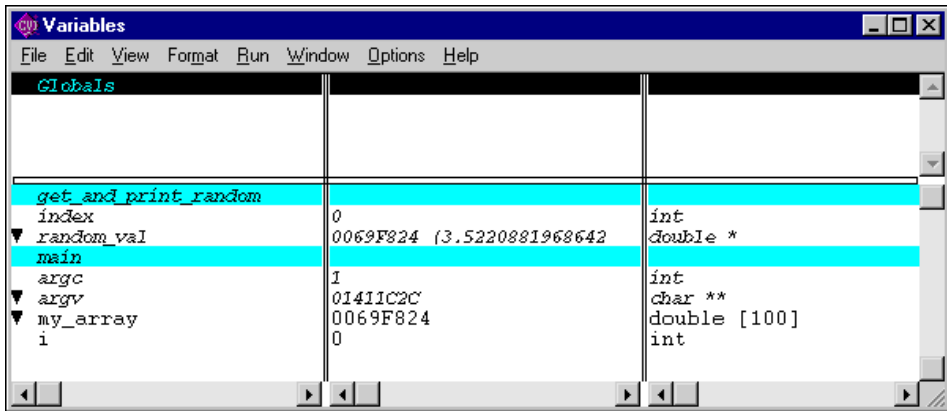


Figure 4-5. Variables Window during Execution of `get_and_print_random`

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.

- Click on the Source window in the background to make it the active window.

This example illustrates the manner in which variables are displayed in the Variables 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 variable contents. The following steps illustrate the use of the Variables window for this purpose.

The program should still be in breakpoint mode with the highlight positioned on the following line inside the function `get_and_print_random`.

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

Perform the following steps.

1. Select **Step Into** from the **Run** menu or press <F8> continuously 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 **View Variable Value** from the **Run** menu, click on the **View Variable Value** toolbar icon, or press <Shift+F7> to automatically display the Variable window with the *i* variable highlighted, shown in Figure 4-6.

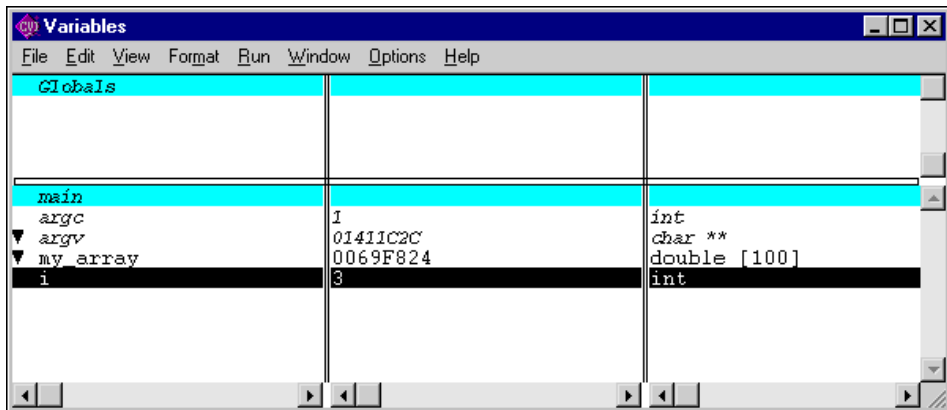


Figure 4-6. Variables Window with *i* Variable Highlighted

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

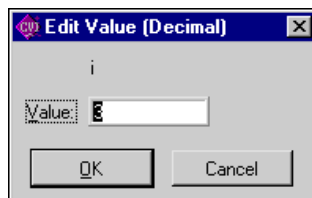


Figure 4-7. Edit Value Dialog Box

4. Enter the value 10 in the dialog box.
5. Press <Enter>. Notice that the value of *i* is now 10.
6. Click on the Source window in the background to make it the active window.
7. Select **Step Into** from the **Run** menu or press <F8> repeatedly.
8. Click on the **Standard Input/Output** window to make it the active window or select **Standard Input/Output** from the **Window** menu.

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

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

The Array Display

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

The program should still be in breakpoint mode. Perform the following steps.

1. Click on the Source window to make it active.
2. Select **Step Into** from the **Run** menu or press <F8> continuously to move the highlight to the following line.

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

3. Click on 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 on the variable name `my_array`. The display shown in Figure 4-8 appears on your screen.

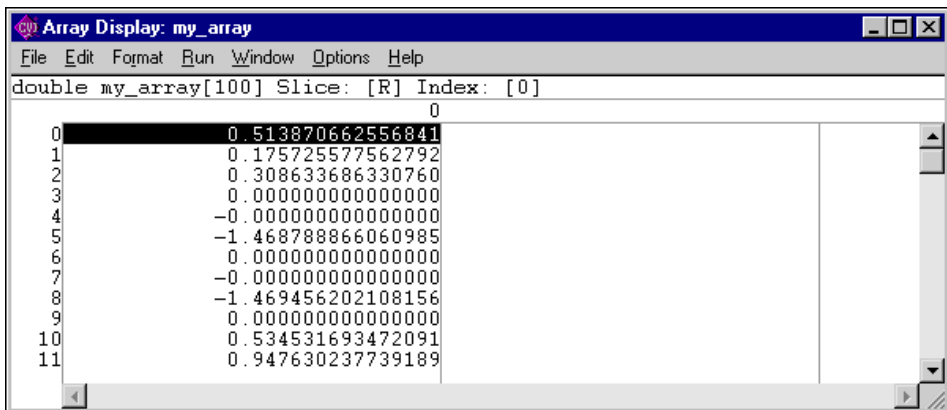


Figure 4-8. Array Display



Note *The actual values in the array may 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 display shown in the preceding illustration is the Array Display. The Array Display shows the values of array elements in tabular format. In this example, 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, with the first element being 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 on 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.

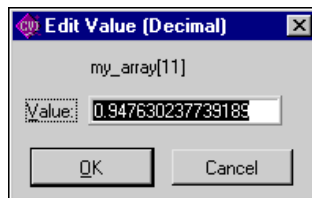


Figure 4-9. Edit Value Dialog Box

2. Enter the value 0.5 and press <Enter>. Notice that the twelfth element of the array is now equal to 0.5.
3. Select **Close** from the **File** menu to remove the Array Display and return to the Variables window.
4. Click on the Source window in the background to make it the active window.
5. Press <F8>, or select **Step Into** from the **Run** menu, to execute the `printf` statement. Notice that the random value printed in the Standard Input/Output window is the value you entered, 0.5.
6. Press <F5>, or select **Continue Execution** from the **Run** menu, to complete program execution.

The String Display

Another useful data display is the String Display. You enter the String Display when you select a string variable from the Variables window. The String Display is similar to the Array Display, except that with the String Display you can view and edit elements of a string. Operations in the String Display are similar to the operations you just performed in the Array Display. For a more detailed description of the String Display window, refer to Chapter 7, *Array and String Display Windows*, in the *LabWindows/CVI User Manual*. An example String Display appears in Figure 4-10.

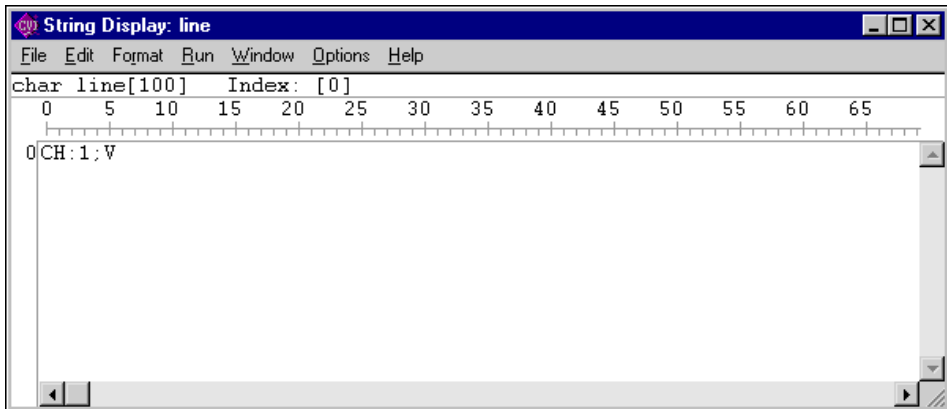


Figure 4-10. String Display

The 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. Return to the Source window if you are still in the Standard Input/Output window.

1. With `sample3.prj` still loaded as the current project, select **Break at First Statement** from the **Run** menu so that a checkmark appears next to it, if it is not already selected from the previous exercise.
2. Select **Run Project** from the **Run** menu, or press <Shift+F5>, to start program execution. Execution will break with the `main` function highlighted.
3. Press <F8> continuously to place the highlight on the following line.

```
void get_and_print_random (index, random_val);
```
4. Select **Variables** from the **Window** menu to open the Variables window, or click on the Variables window if it is already active in the background.

5. Click on the `random_val` variable or press the <Down> arrow key until the `random_val` variable is highlighted.
6. Select **Add Watch Expression** from the **Options** menu to indicate that you wish to assign the `random_val` variable to the Watch window. The dialog box shown in Figure 4-11 will appear.

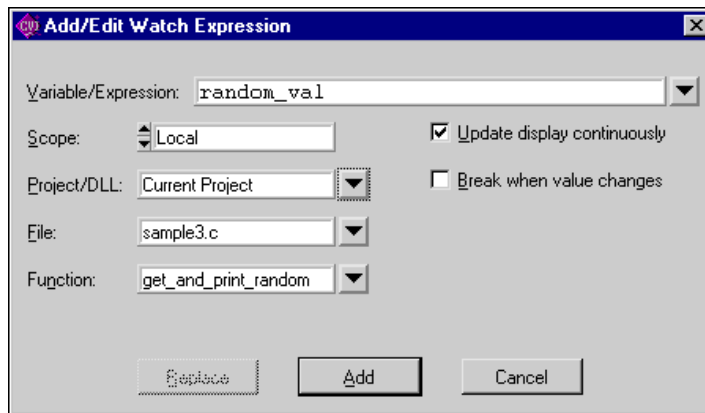


Figure 4-11. Add/Edit Watch Expression Dialog Box

7. Click in the check box next to **Update Display Continuously**, if it is not already checked. Click on the **Add** button.
8. Click on 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 **Continue Execution** from the **Run** menu to complete program execution. As the program continues running, you can view the value of the `random_val` variable changing dynamically in the Watch window.
10. When program execution is complete, close all windows except the Project window.

This session concludes the first part of the tutorial session in which you became acquainted with the LabWindows/CVI programming environment. In the next part of the tutorial, Chapters 5, 6, 7, and 8, you will build an actual project in LabWindows/CVI.

Part II

Tutorial—Building an Application in LabWindows/CVI

Building a Graphical User Interface

This chapter contains instructions for building a project consisting of a Graphical User Interface and a C source file.

This chapter begins the second part of the tutorial sessions in which you will build an actual project in LabWindows/CVI. These sessions will be included in Chapters 5, 6, 7, and 8. In Chapter 9, you will find additional exercises to practice and expand on what you have learned.

In the first tutorial session, you executed a sample program that was controlled with a graphical user interface (GUI) developed in the User Interface Editor. Throughout the remaining sessions of this tutorial, you will develop a simple project consisting of a GUI controlled by a C source file. In this session, you will learn to design a user interface with the User Interface Editor. In subsequent tutorial sessions, you will 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 program. A user interface is comprised of a collection of objects such as menu bars, panels, controls, and pop-up menus. In the Chapter 6 tutorial session, you will be introduced to the User Interface Library, which includes a set of functions for controlling the interface programmatically.

Setting Up

Perform the following steps to set up this session.

1. Select **New** from the **File** menu. Select `Project (*.prj)` to unload the existing `sample3.prj` project. A dialog box will come up on your screen asking if you are sure you want to unload the current project. Press `<Enter>` or click on **Yes**.
2. The dialog box shown in Figure 5-1 should appear. Click on **OK** to maintain the current configuration of the LabWindows/CVI environment for the new project.

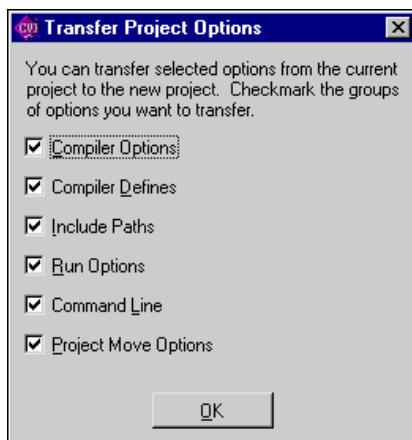


Figure 5-1. Transfer Project Options Dialog Box

3. Close all of the windows visible except the Project window.

The User Interface Editor

The User Interface Editor is an interactive drop-and-drag editor for designing custom graphical user interfaces (GUIs). You can select a number of different controls from the **Create** menu and position them on your panels. Each control can be customized 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 have designed your user interface in the User Interface Editor window, you can begin writing your C source code to control the GUI. You must give a name to each panel, menu, and control on your user interface that can be used in the C source code to differentiate the controls on the GUI. You can also assign a function name to controls on your user interface that will be called automatically whenever you operate that control during program execution. The Constant Name and Callback Function associated with a particular control are assigned within the edit dialog box for the control in the User Interface Editor.

Once you have completed a user interface and saved it as a resource file (`.uir`), LabWindows/CVI automatically generates an include file defining all of the constants and callback functions you have assigned.

CodeBuilder

After you have completed your `.uir` file, LabWindows/CVI can help get you started on your source file with the CodeBuilder utility. CodeBuilder will automatically create a source file based on the callback functions specified in your `.uir` file.

The Sample Project

In the next few sessions of this tutorial, you will follow instructions to build a sample program that acquires and displays a waveform on a GUI. The development process will be as follows.

1. Create a User Interface in the User Interface Editor (Chapter 5).
2. Generate a shell program source file using CodeBuilder (Chapter 5).
3. Add to the C source code to generate and display the waveform (Chapter 6).
4. Develop your own callback function to compute the mean value of the waveform (Chapter 7).
5. Incorporate the use of an instrument driver in the project to simulate data acquisition (Chapter 8).

Building a User Interface Resource (`.uir`) File

The following steps teach you how to build a user interface, an additional command button, and a graph control to the user interface of the sample project.

Step 1: Opening a .uir File

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

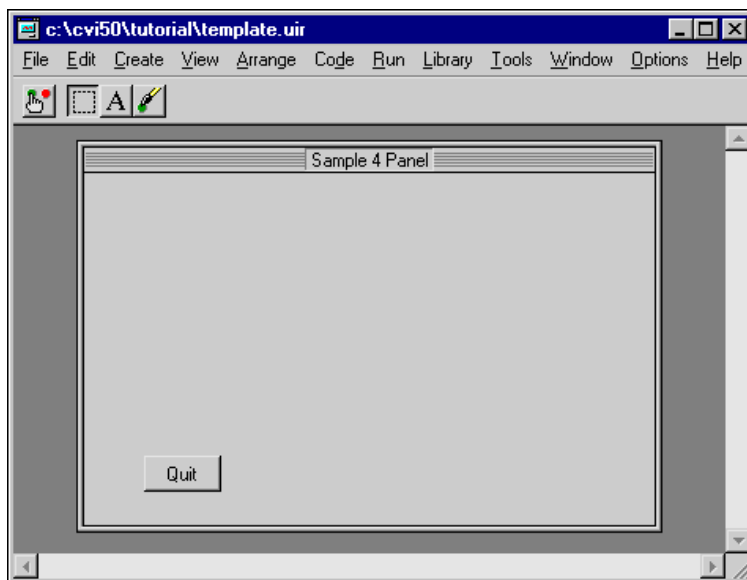


Figure 5-2. User Interface Editor

The **Edit** menu contains selections that can cut, copy, paste, align, and space user interface controls in the editor.

The **Create** menu contains selections for placing user interface objects, such as numeric controls, LEDs, command buttons, toggle switches, graphs, and strip charts, onto your user interface file.

- 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 will add the button and graph control in the steps that follow.

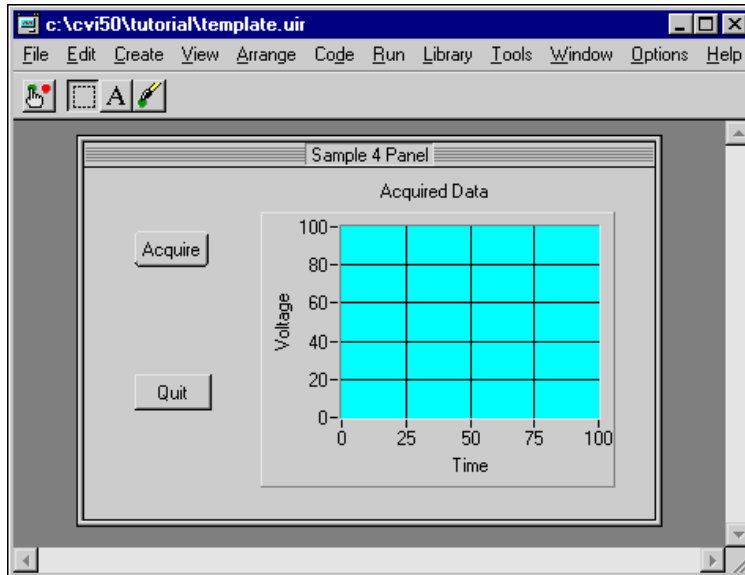


Figure 5-3. Sample User Interface

Step 2: Adding a Command Button

- Select **Command Button** from the **Create** menu and choose one of the button styles shown in the pull-right menu. You should see a button labeled **OK** on the panel.
- Use the mouse to position the button on the panel. If you do not have a mouse, press the <Tab> key until the **OK** button is highlighted, and then use the arrow keys to position the button.

- To edit the button attributes, double-click on the button or press <Enter>. A dialog box entitled Edit Command Button should appear as shown in Figure 5-4.

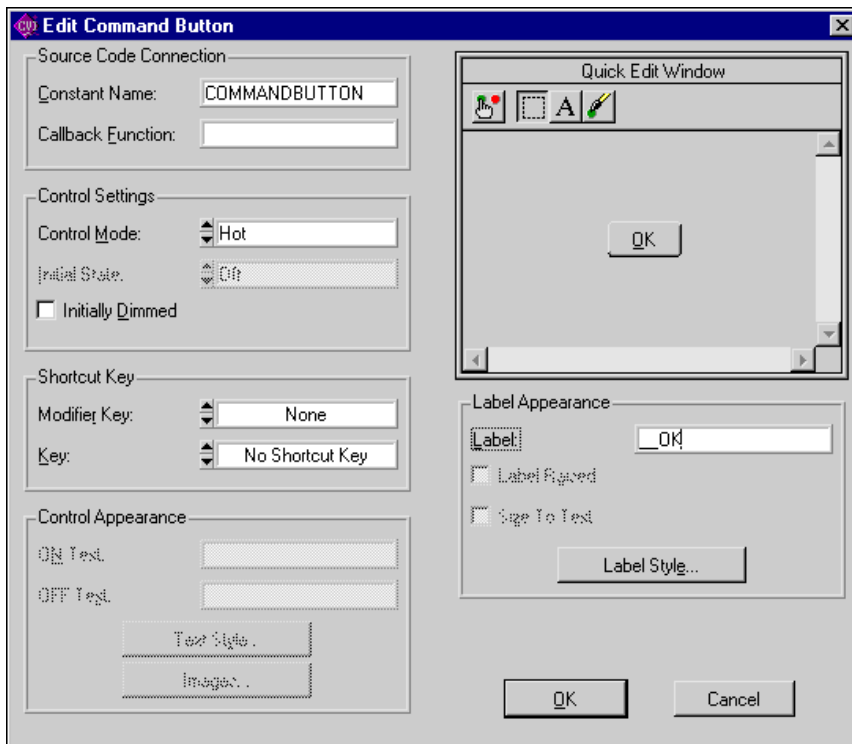


Figure 5-4. Edit Command Button Dialog Box

- First, you must assign a constant name to the button. Your C source code will use this constant name to communicate with the button. LabWindows/CVI creates a default constant name for you. You can customize your .uir file with your own Constant Names. Type ACQUIRE in the Constant Name input within the Source Code Connection section of the dialog box. (Be sure to use all capital letters.)
- Next, you must assign a function name to be called whenever a user clicks on the **Acquire** button. Type AcquireData in the Callback Function input. In the next chapter you will 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.

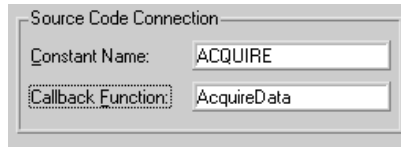


Figure 5-5. Source Code Connection Section of Edit Command Button Dialog Box

6. Press the <Tab> key five times or use the mouse to highlight **OK** in the Label input within the Label Appearance section of the dialog box.
7. Change the label on the command button by typing `Acquire` in the Label input.
8. (Optional) Further customize the appearance of the label by clicking on the **Label Style** button to open another dialog box. Click on the **OK** button when you are finished.
9. Click on the **OK** button at the bottom of the Edit Command Button dialog box or press <Enter>. Your user interface should look similar to the one shown in Figure 5-6.

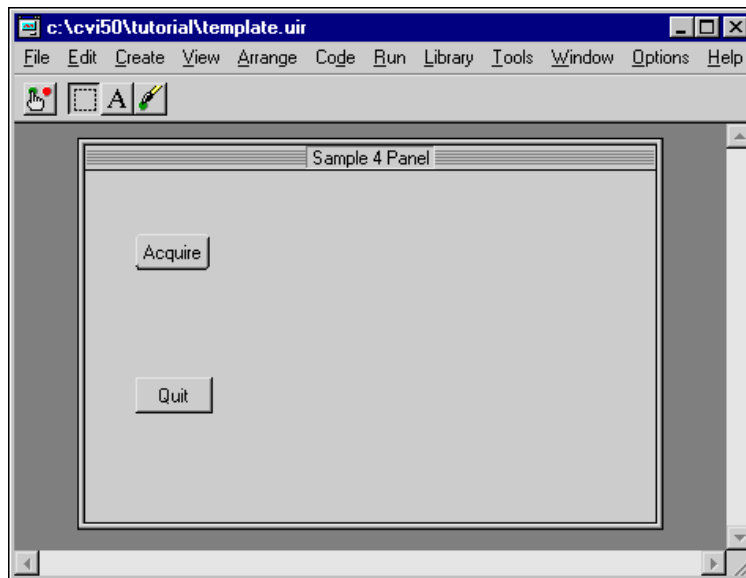


Figure 5-6. User Interface with Two Buttons

Step 3: Adding a Graph Control to the User Interface

1. Pull-down the **Create** menu and select the Graph control from the **Graph** submenu. A graph control named `Untitled Control` should appear on your user interface.

2. Position the graph with the mouse. Size the graph by clicking and dragging one of the corners.
3. Double-click on the graph control to display the Edit Graph dialog box for customizing the graph attributes.
4. Type WAVEFORM in the Constant Name input within the Source Code Connection section of the dialog box. Be sure to use all capital letters.



Note *Because the graph will only be used as an indicator for displaying a waveform in this program, you do not need to assign a callback function to the graph control. Callback functions are only necessary when the operation of the control initiates an action or acts as an input. Indicators generally do not require callback functions.*

5. Type Acquired Data in the Label input within the Label Appearance section of the dialog box. The dialog box should look exactly as shown in Figure 5-7.

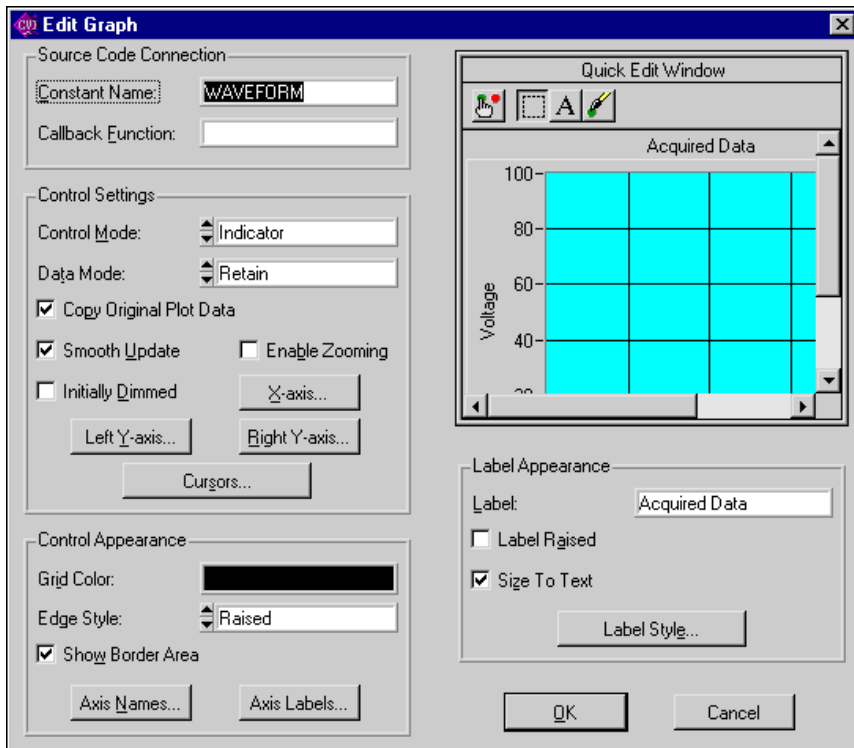


Figure 5-7. Edit Graph Dialog Box

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.

- After you have finished setting the graph attributes, click on the **OK** button at the bottom of the Edit Graph dialog box to close the dialog box.

Step 4: Saving the .uir File

- Your completed user interface should look like the one shown in Figure 5-8.

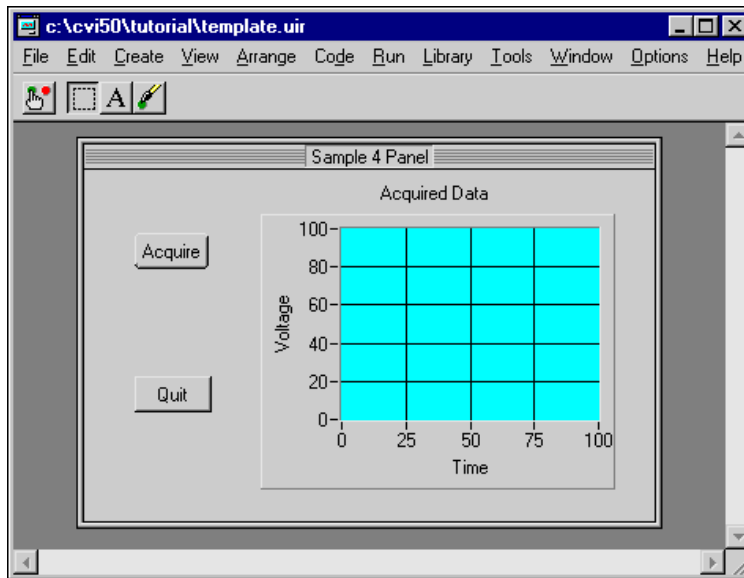
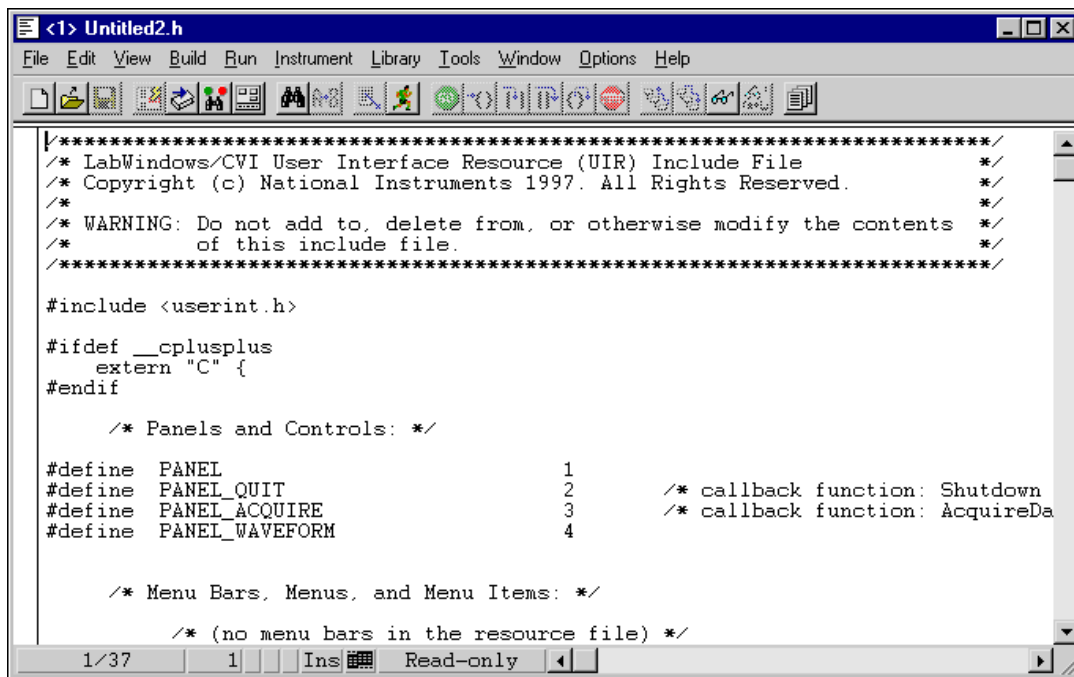


Figure 5-8. Completed User Interface

- Select **Save As** from the **File** menu to save the .uir file with the new controls added.
- Type `sample4` in the File Name input in the Save File As dialog box. Click on the **Save** button to close the File Save dialog box.

4. Select **Preview User Interface Header File** from the **View** menu to view the header file that LabWindows/CVI has automatically created. It should appear as shown in Figure 5-9.



```

<1> Untitled2.h
File Edit View Build Run Instrument Library Tools Window Options Help
|*****|
/* LabWindows/CVI User Interface Resource (UIR) Include File */
/* Copyright (c) National Instruments 1997. All Rights Reserved. */
/*
/* WARNING: Do not add to, delete from, or otherwise modify the contents
/* of this include file.
/*
|*****|

#include <userint.h>

#ifdef __cplusplus
extern "C" {
#endif

    /* Panels and Controls: */

#define PANEL                1
#define PANEL_QUIT          2      /* callback function: Shutdown
#define PANEL_ACQUIRE      3      /* callback function: AcquireDa
#define PANEL_WAVEFORM      4

    /* Menu Bars, Menus, and Menu Items: */

    /* (no menu bars in the resource file) */
1/37  1  Ins  Read-only

```

Figure 5-9. User Interface Header File

5. Close the header file before continuing by selecting **Close** from the **File** menu. You do not need to save it.

Step 5: Generating the Program Shell with CodeBuilder

Now that you have created a GUI, CodeBuilder can help you get a jump start on development by automatically generating the start of your source code.

1. First, you must specify the events to which your program will respond. Select **Default Control Events** from the **Preferences** selection of the **Code** menu. The dialog box shown in Figure 5-10 appears.

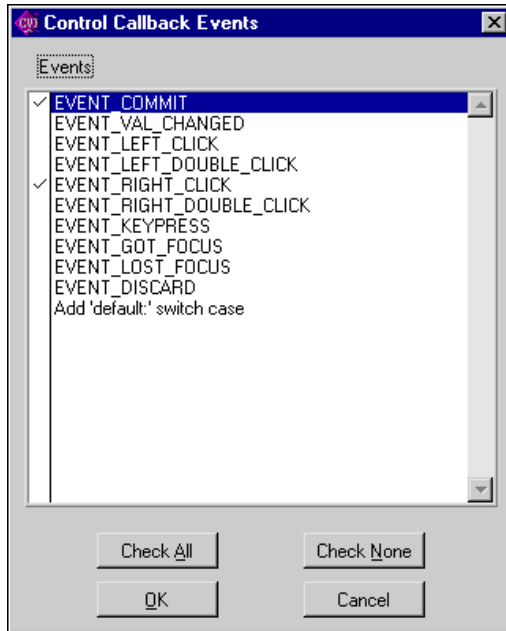


Figure 5-10. Control Callback Events Dialog Box

2. Later in this tutorial, you will develop code to display help information when a user right-clicks on a GUI control. To do this, you must select `EVENT_RIGHT_CLICK` from the Control Callback Events dialog box so that a checkmark appears next to it. Your program will respond to two events: a commit event (left-click or <Enter>) that generates data and plots it on the graph, and a right-click that displays help. Select **OK**.

3. Select **Generate»All Code** from the **Code** menu to display the dialog box in Figure 5-11.

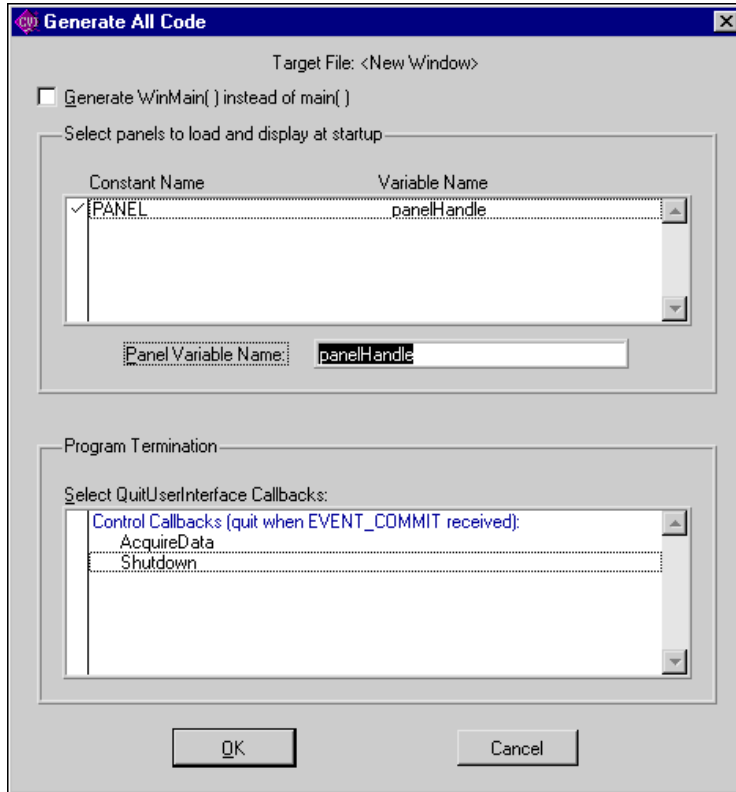


Figure 5-11. Generate All Code Dialog Box

4. You must specify some options from the CodeBuilder dialog. First you must decide which panels you want to display at program startup. For this program, you only have one panel in your `.uir` file.



Note For this exercise, make sure that the **Panel Variable Name** is `panelHandle`.

5. The lower half of the dialog shows a list of callback functions in your `.uir` file. You can select a function from this list that will cause the program to terminate execution. Select the **Shutdown** function in the dialog so that a checkmark appears next to it.

6. Select the **OK** button. This will trigger CodeBuilder to build the source code for your program. A new Source window should appear 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 ();
    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 **Save** from the **File** menu in the Source Window or click on the **Save** toolbar icon and save the source file with the name `sample4.c`.

This concludes the Chapter 5 tutorial session.

Complete the next session to complete the C source code for displaying a waveform on the graph.

Using Function Panels and the Libraries

In this session, you will use 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 the last session (Chapter 5). If you have not completed Chapter 5, go back and do so now.

Setting Up

If you did not directly proceed from Chapter 5, follow these steps to set up LabWindows/CVI so that you can complete this tutorial session.

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

Analyzing the Source Code

The source code for the `sample4` program is incomplete. In this session, you will 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 be writing similar functions in the future for your own LabWindows/CVI programs.

The main Function

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

```
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 ();
    return 0;
}
```

Before you can display or operate the user interface that you created in the last session, 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.

The AcquireData Function

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

The Shutdown Function

The `Shutdown` function automatically executes whenever you click on the **Quit** button from 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 will take place in the `AcquireData` function. When a user clicks on the **Acquire** button, the program will generate 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 has utilities to help you generate code for common C constructs such as for loops, while loops, and switch statements. Select **Insert Construct** from the **Edit** menu and choose **For Loop** to display the dialog box shown in Figure 6-1.

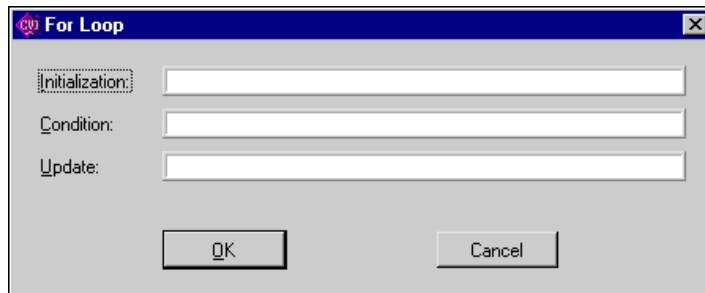


Figure 6-1. For Loop Dialog Box

4. Enter the following values in the For Loop dialog.
Initialization: `i=0`
Condition: `i<100`
Update: `i++`
Select **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. Pull down the **Library** menu and select **User Interface** to display the dialog box shown in Figure 6-2.

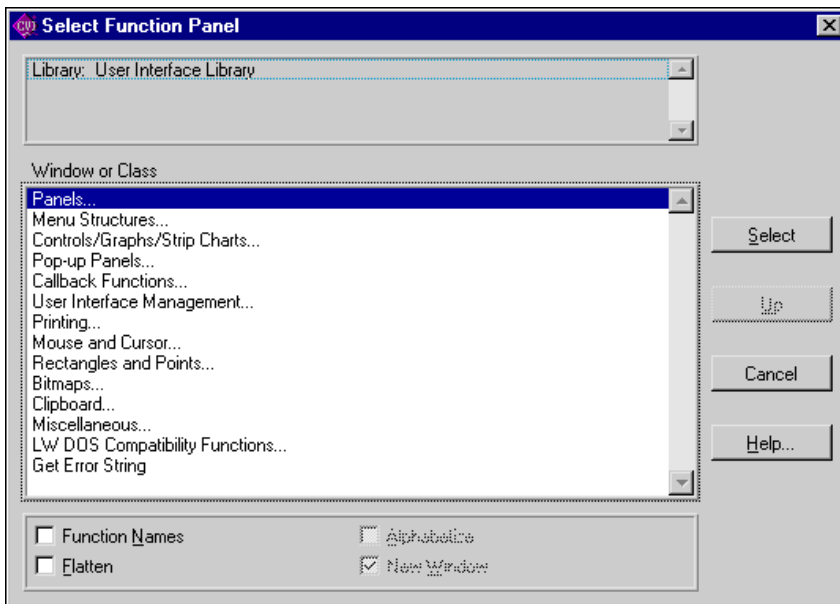


Figure 6-2. Select Function Panel Dialog Box

3. Use the <Down> arrow key to highlight the **Controls/Graphs/Strip Charts** selection and press <Enter>.
4. Use the <Down> arrow key to 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 of the LabWindows/CVI functions related to displaying or operating data on graphs and strip charts, as shown in Figure 6-3.

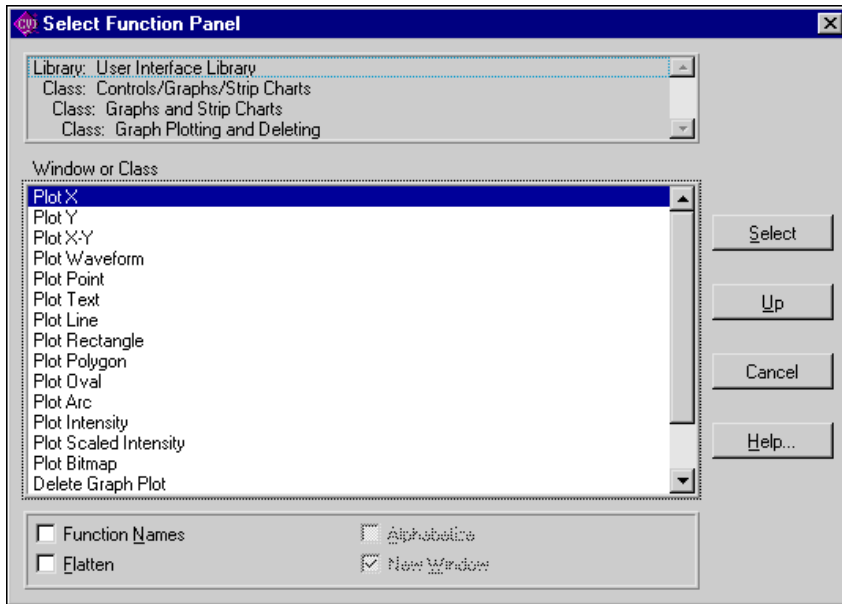


Figure 6-3. Graph Plotting and Deleting Functions

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

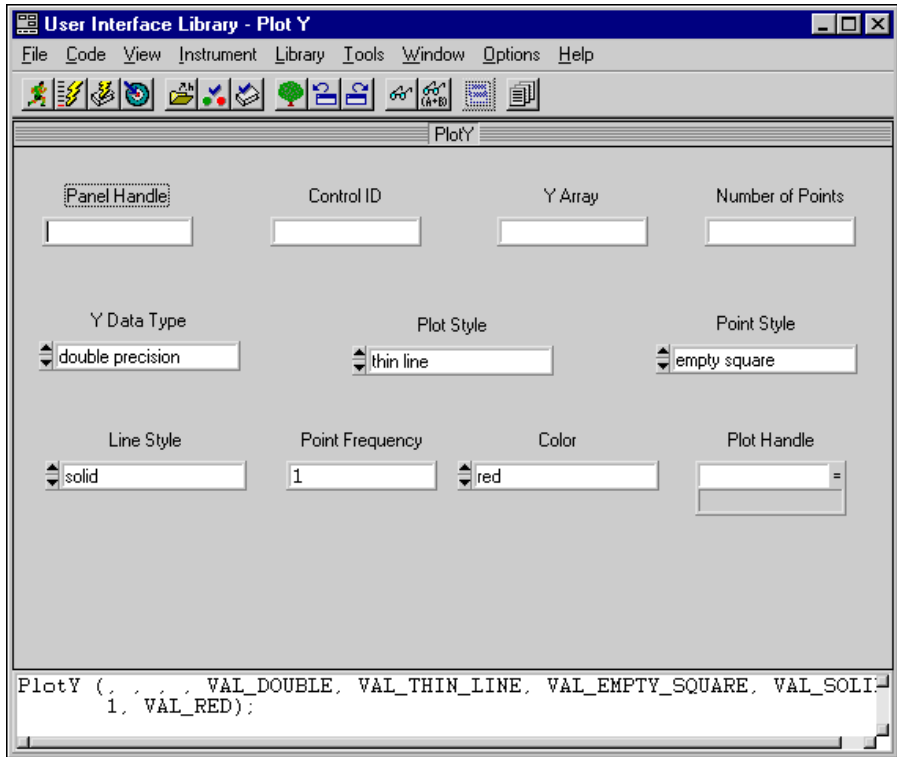


Figure 6-4. Plot Y Function Panel

Building the PlotY Function Call Syntax

In this session, you will use the `PlotY` function panel shown in the previous illustration 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 **Select Variable** from the **Code** menu. 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 of the Constant Names in the `.uir` file you are working on, select **Select UI Constant** from the **Code** menu. A list of all constant names in the `.uir` file should appear as shown in Figure 6-5.

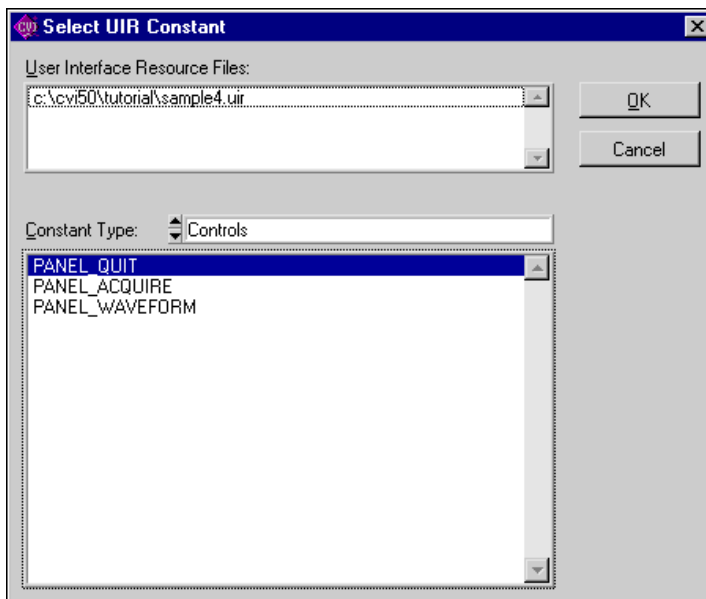


Figure 6-5. Select UIR Constant Dialog Box

Select `PANEL_WAVEFORM` and click on the **OK** button to continue.

3. Type `datapoints` into the Y Array control. This name indicates which array in memory will be displayed on the graph.
4. Type `100` into the Number of Points control. This number indicates how many elements in the array are to be plotted.

- You have completed generating the source code using the `PlotY` function panel, which should look like Figure 6-6.

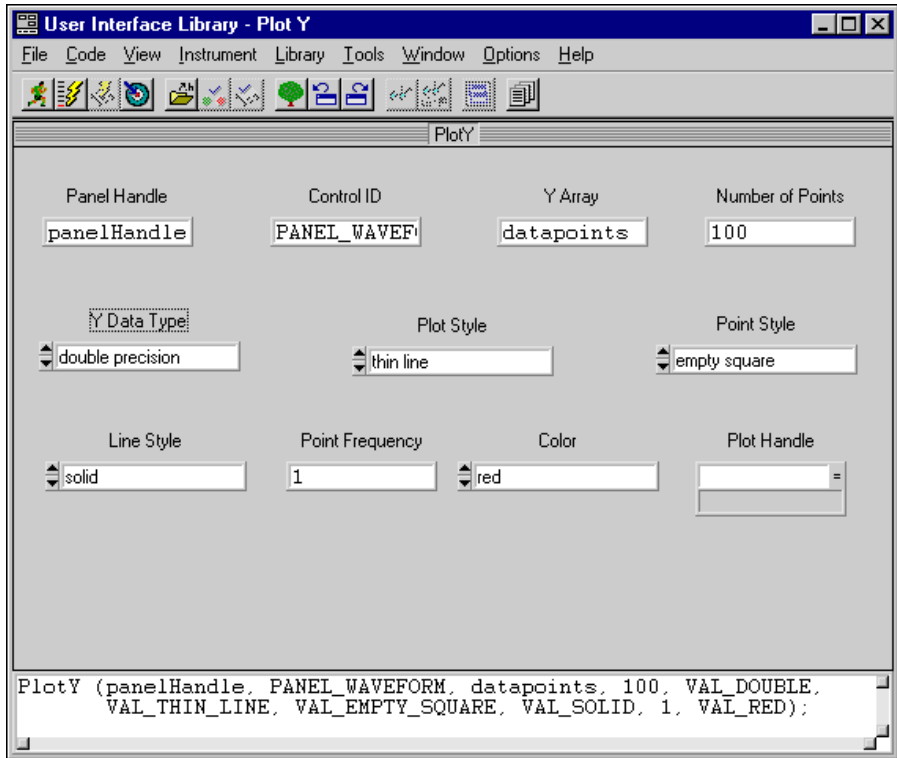


Figure 6-6. Completed Plot Y Function Panel

- Select **Set Target File** from the **Code** menu 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 on **OK**.
- Select **Insert Function Call** from the **Code** menu to paste the `PlotY` function call into your source code.

- Close the function panel by selecting **Close** from the **File** menu.

Your program should match the source code shown in Figure 6-7.

```

return 0;
}

int CVICALLBACK AcquireData (int panel, int control, int event,
void *callbackData, int eventData1, int eventData2)
{
switch (event) {
case EVENT_COMMIT:
for (i=0; i<100; i++)
datapoints[i] = rand()/32767.0;
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;
}

```

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

- Select **Save** from the **File** menu, or select **Save As...** from the **File** menu and enter `sample4.c` in the dialog box and press <Enter>.

Constructing the Project

In Chapter 2, In the first tutorial session, you were introduced to the Project window. The Project window contains a list of files that make up the program you are developing. In this case, you have three files that should be listed in your project.

- `sample4.uir`—The user interface file that you built in the last session.
- `sample4.h`—The include file that was generated automatically when you saved the `.uir` file in the last session. The include file declares all of the constant names and callback functions that you assigned in the User Interface Editor.
- `sample4.c`—The source file created in this session.

Follow these steps to construct your project.

1. Close all windows except the Project window (`untitled1.prj`).
2. Select **Add File to Project** from the **Edit** menu. Choose `All Files (*.*)` as the file type. 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. Checkmarks will appear next to these files, and the filenames will appear in the Selected Files dialog box shown in Figure 6-8.

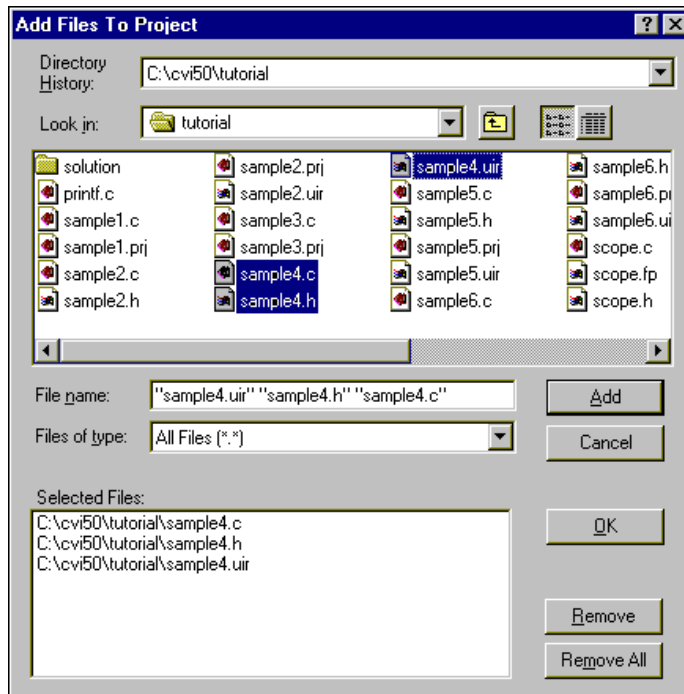


Figure 6-8. Add Files to Project Dialog Box

4. Click on **OK**.
5. Select **Save** from the **File** menu.
6. Enter `sample4` in the File Name input. Press `<Enter>` or click on **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 Project** from the **Run** menu to execute the code.

During the compile process, LabWindows/CVI will recognize that your program is missing the `ansi_c.h` include statement. Choose **Yes** to add this include file to your program. When prompted next, 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 on the **Acquire** button, 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 on the **Quit** button, the event information generated by the mouse is passed directly to the `Shutdown` function, which halts the program.

This concludes the Chapter 6 session of the tutorial. In the next session you will learn to add simple analysis capability to your program.

Adding Analysis to Your Program

In the previous session, 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 session, you will add 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.

The objectives of this session 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 session builds on the concepts that you learned in the previous session. If you have not completed Chapter 6, go back and do so now.

Setting Up

1. Close all Windows except the Project window.
2. Select **Open** from the **File** menu and choose `Project (*.prj)` as the file type. Load `sample5.prj`.
3. Select **Run Project** from the **Run** menu to verify the operation of the program. The `sample5` file should match the project you completed in the Chapter 6 tutorial session.

Goals of Session

In this session, you will 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 should have completed previous sessions in this tutorial on the User Interface Editor (Chapter 5) and code generation (Chapter 6 tutorial), the instructions for performing these tasks will be less explicit in this session.

Modifying the User Interface

Your first task is to modify the user interface that you built in the Chapter 5 tutorial. 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 may be unnecessary because CodeBuilder's default preference setting is to append generated code to the bottom of the file. This location will be used by CodeBuilder for the new callback function that will be generated later.

- 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.

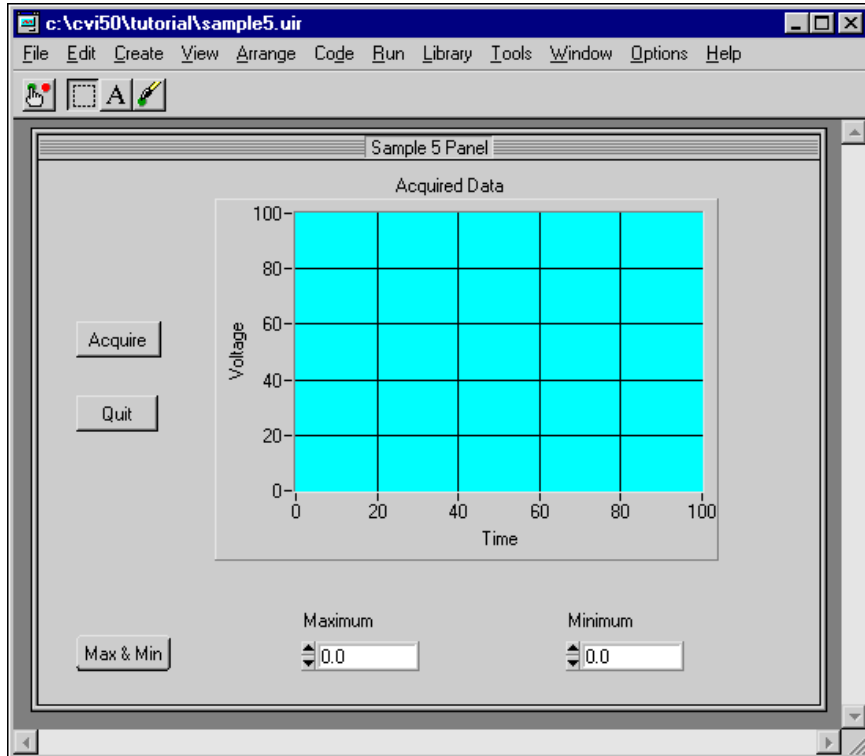
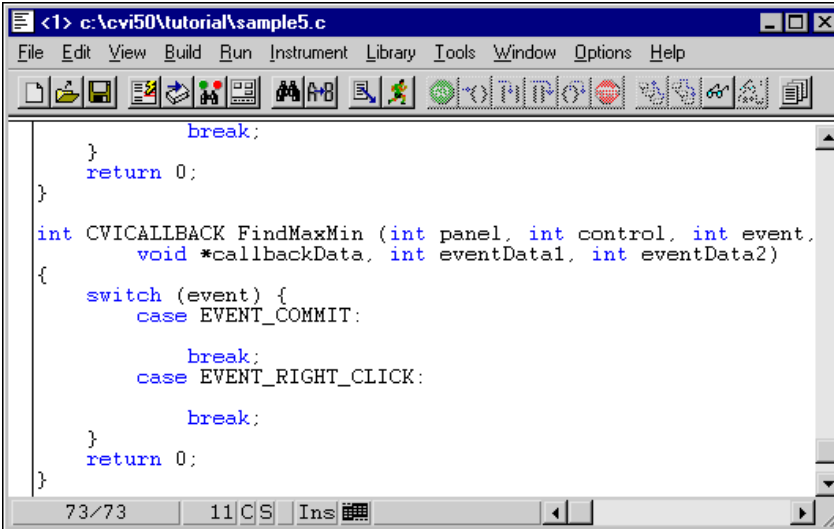


Figure 7-1. Sample User Interface

- Select **Command Button** from the **Create** menu and choose a command button from the control palette.
- Double-click on the **Command** button to bring up the 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 on the **Max & Min** command button and select **Generate Control Callback** from the popup menu.

A screenshot of the CodeBuilder IDE window. The title bar shows the file path 'c:\cvi50\tutorial\sample5.c'. The menu bar includes 'File', 'Edit', 'View', 'Build', 'Run', 'Instrument', 'Library', 'Tools', 'Window', 'Options', and 'Help'. The toolbar contains various icons for file operations and development tools. The main text area displays the following C code:

```
        break;
    }
    return 0;
}

int CVICALLBACK FindMaxMin (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;
}
```

The status bar at the bottom shows '73/73', '11|C|S|', and 'Ins'.

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 have completed updating the user interface for `sample5`, you will add to the `FindMaxMin` callback function to compute and display the maximum and minimum values of the array.

Select **Numeric** from the **Create** menu 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 bring up 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 **Numeric** from the **Create** menu and choose the numeric control from the upper left corner of the control palette, as you did in step 4.
8. Double-click on the numeric control to bring up the dialog box. Enter the following information into the dialog box.

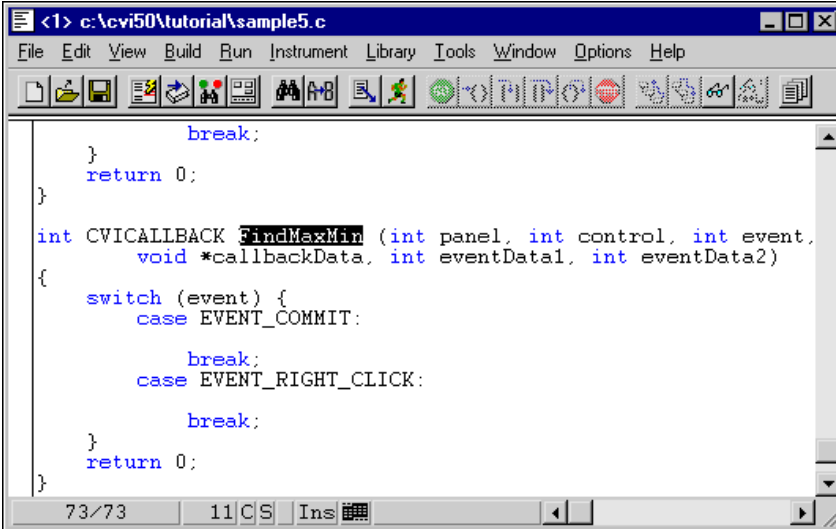
Constant Name:	MIN
Control Mode:	Indicator
Label:	Minimum

9. Position the two new controls on the user interface to match those in the illustration of the Sample 5 Panel shown previously.
10. Select **Save** from the **File** menu 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, your next task is 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 on the **Max & Min** button in the User Interface Editor to display the CodeBuilder popup menu again. Select **View Control Callback**. CodeBuilder displays the `sample5.c` source file with the `FindMaxMin` callback function highlighted. The callback function should appear as shown in Figure 7-4.



```

<1> c:\cvi50\tutorial\sample5.c
File Edit View Build Run Instrument Library Tools Window Options Help

}
    break;
}
return 0;
}

int CVICALLBACK FindMaxMin (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;
}

73/73 11 C S Ins

```

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 if statement will be executed when your program is running and you click on 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 will 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 will show either the **Analysis library** or the **Advanced Analysis library**.

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

Input Array:	<code>datapoints</code>
Number of Elements:	<code>100</code>
Maximum Value:	<code>max</code>
Maximum Index:	<code>max_index</code>
Minimum Value:	<code>min</code>
Minimum Index:	<code>min_index</code>

- 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 **Declare Variable** from the **Code** menu. Set the check box to **Add declaration to current block**. This inserts a line of code to declare the `max` variable within the `FindMaxMin` callback function. Click on the **OK** button to continue.
- Notice that LabWindows/CVI automatically inserts an ampersand “&” before the `max` variable so that it is properly passed by reference to the function.
- Repeat the **Declare Variable** steps for the **Maximum Index**, **Minimum Value**, and **Minimum Index** controls on the `MaxMin1D` function panel.
- Insert the `MaxMin1D` function call into your source code by selecting **Insert Function Call** from the **Code** menu. Close the `MaxMin1D` function panel. You should see the `MaxMin1D` function inside the `EVENT_COMMIT` case statement within the `FindMaxMin` callback function.
- Display the `SetCtrlVal` function panel by selecting **Library»User Interface»Controls/Graphs/Strip Charts»General Functions»Set Control Value**.
- 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 at the array in the **Maximum** numeric display.

Panel Handle:	<code>panelHandle</code>
Control ID:	<code>PANEL_MAX</code>
Value:	<code>max</code>
- Insert the `SetCtrlVal` function call into your source code by selecting **Insert Function Call** from the **Code** menu. You will see the `SetCtrlVal` code entered on the line after the function call to `FindMaxMin` in your source code file.
- Now display the `SetCtrlVal` function panel again by selecting **Library»User Interface»Controls/Graphs/Strip Charts»General Functions»Set Control Value**.

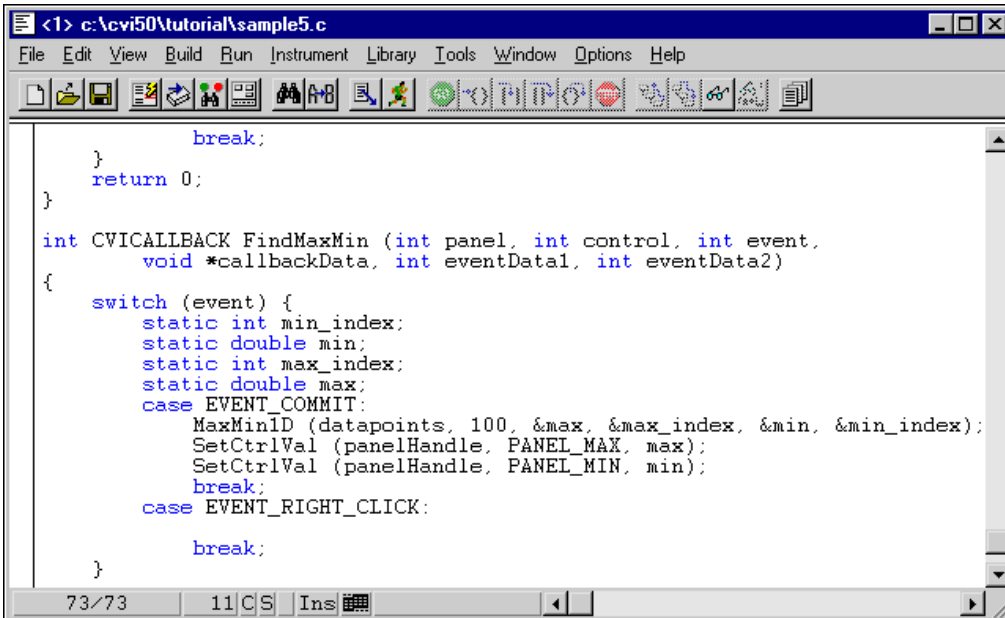
- The `SetCtrlVal` function sets the value of a control on your user interface. Enter the following information into the function panel controls to display the minimum value of the array in the numeric display called Minimum:

Panel Handle: `panelHandle`

Control ID: `PANEL_MIN`

Value: `min`

- Insert the `SetCtrlVal` function call into your source code by selecting **Insert Function Call** from the **Code** menu. Close the `SetCtrlVal` function panel.
- Your source code should match the code shown in Figure 7-5.



```

    break;
}
return 0;
}

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

```

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 will be called when your program is running and you click on the **Max & Min** command button. When you click on the **Max & Min** command button, 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 have written the function to find the Max & Min values and display them only when the `COMMIT` event is sensed. For more practice with user interface events, work exercise number 5 in Chapter 9.

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

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

The next tutorial session, Chapter 8, will introduce you to the instrument drivers in LabWindows/CVI. In Chapter 8, you will use an instrument driver to acquire data, instead of generating a random array.

Using an Instrument Driver

In this session you will 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 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.

Setting Up

This session builds on the program that you created in Chapter 7. If you have not completed that session, please do so now. Before beginning this example, follow these steps to set up the screen display.

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

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 **Load** from the **Instrument** menu in the Project window. The Load Instrument dialog box appears.
2. Press <Tab> repeatedly to select the large list box containing directory and file names 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.

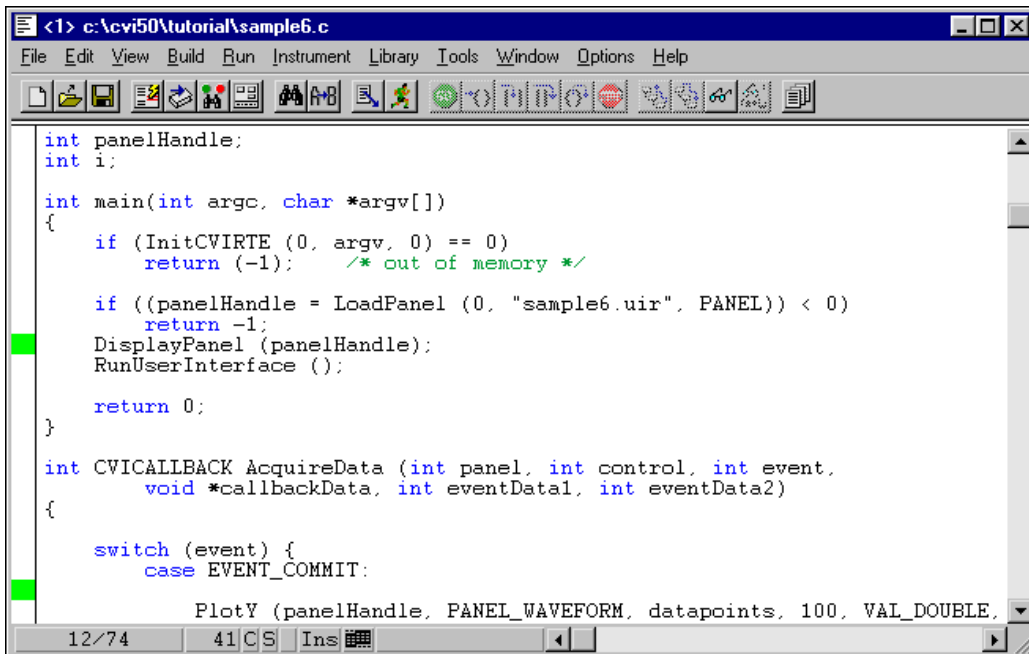


Figure 8-1. sample6.c Source File with Tags

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.

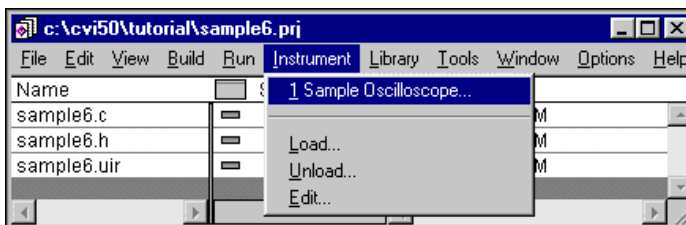


Figure 8-2. Sample Oscilloscope on Instrument Menu

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 the **Sample Oscilloscope** item from the **Instrument** menu. The dialog box shown in Figure 8-3 appears.

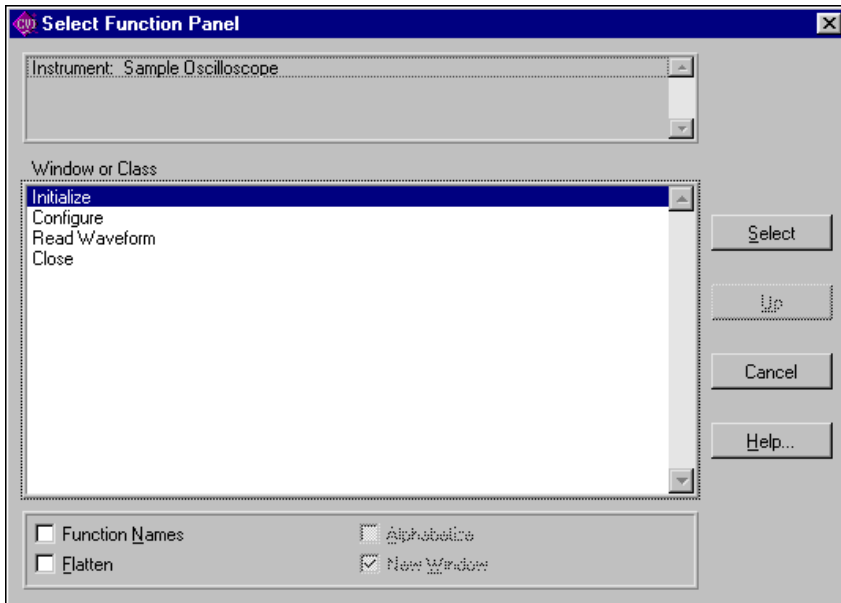


Figure 8-3. Sample Oscilloscope Functions

This module contains the following functions:

- Initialize
- Configure
- Read Waveform
- Close

You will use these functions to acquire a waveform in the sample program you develop throughout this tutorial. Leave this dialog box open. You will 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, you learned how to use function panels to generate code and insert it 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 session you will learn how to execute function panels first, 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.

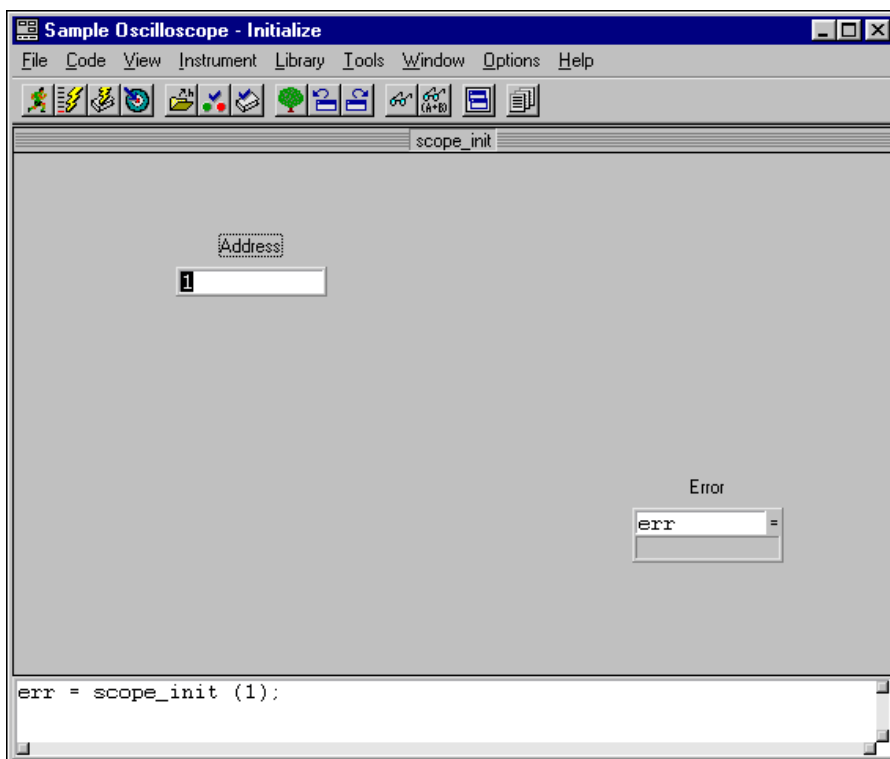


Figure 8-4. Initialize Function Panel

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 **Window** from the **Help** menu. 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 **Declare Variable** from the **Code** menu. Be sure to click on the following checkbox items: Execute declaration and Add declaration to the top of target file. Click on **OK**.
4. Select **Run Function Panel** from the **Code** menu. Click on **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 **Program** window and remove the function panel from the screen.

1. Select **Insert Function Call** from the **Code** menu, or press <Ctrl-I>, to copy the generated code to the Source window.
2. You might have 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 have not set it. Select `sample6.c` from the dialog box.
3. Select **Close** from the **File** menu.

The function call to initialize the instrument driver will appear above the `DisplayPanel` function call in the Source window.

```
[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

Sample Oscilloscope from the **Instrument** menu. Select **Configure** from the dialog box. The function panel shown in Figure 8-5 appears on your screen.

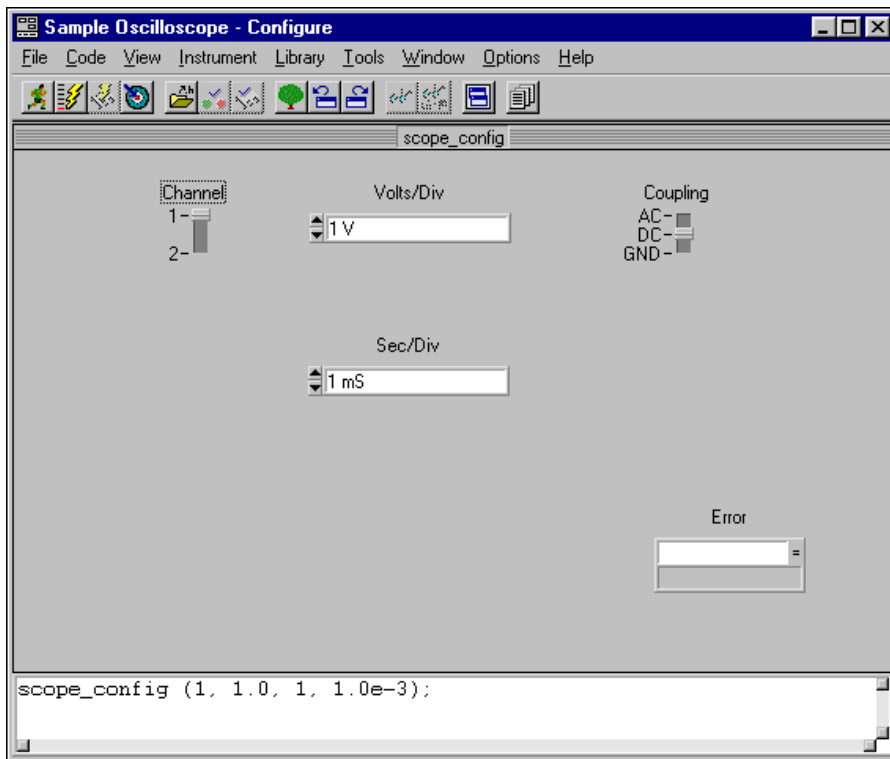


Figure 8-5. Configure Function Panel

The `Configure` panel 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 will 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 **Run Function Panel** from the **Code** menu to execute the function panel.

If the Error control does not display a zero (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 **Run Function Panel** from the panel. Thus, you could interactively program the instrument and verify the operation of the instrument driver functions.

3. Select **Insert Function Call** from the **Code** menu 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 session.

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 **Read Waveform** from the **Sample Oscilloscope** instrument driver in the **Instrument** menu. The Read Waveform function panel appears as shown in Figure 8-6.

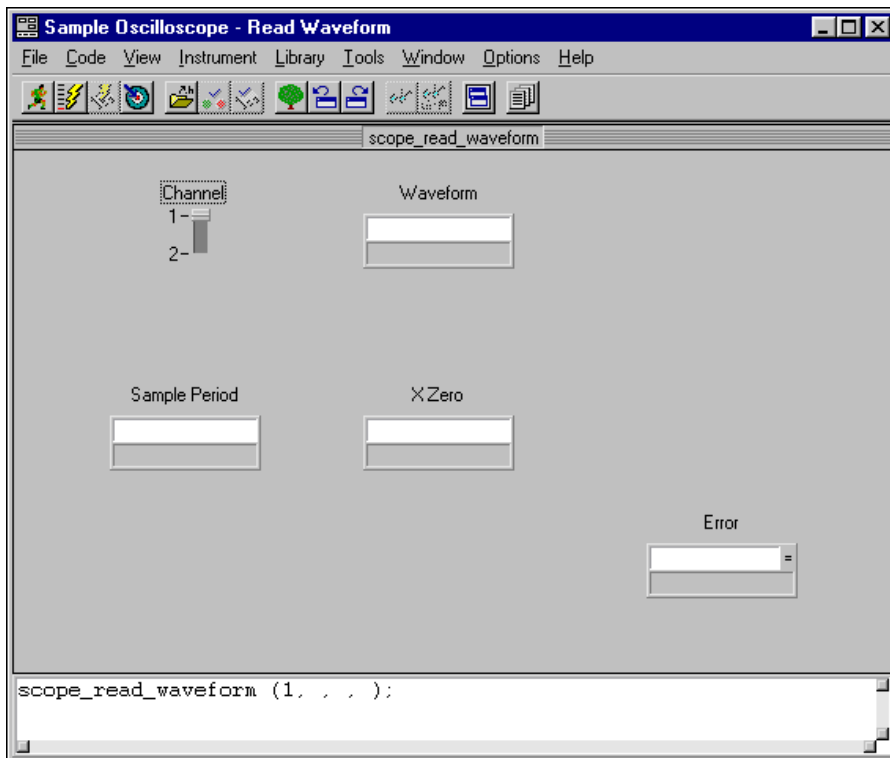


Figure 8-6. Read Waveform Function Panel

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 **Declare Variable** from the **Code** menu. A dialog box appears with the `datapoints` variable automatically entered in the Variable Name input.
4. Press <Tab> to select the Number of Elements text box.
5. Enter `100` into the Number of Elements text box.
6. Press <Tab> twice, so that the Add declaration to the top of target file `sample6.c` option is highlighted.
7. Press <Space> to place a checkmark in the Add Declaration checkbox, if it is not already present.
8. The dialog box appears as shown in Figure 8-7.

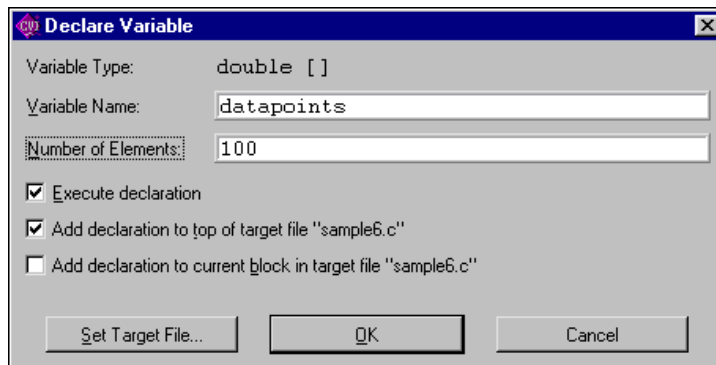


Figure 8-7. Declare Variable Dialog Box

9. Press <Enter> to declare the `datapoints` array.

Reading the Waveform

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

1. Press <Tab> repeatedly to select the Sample Period control or click on the Sample Period control.
2. Select **Declare Variable** from the **Code** menu.
3. Enter the variable name `delta_t` in the Variable Name input box and press <Enter>.
4. Press <Tab> to highlight the X Zero control.
5. Select **Declare Variable** from the **Code** menu.
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 **Run Function Panel** from the **Code** menu to execute the function panel. Save changes before running. If the Error control does not show 0, 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 Variable 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 **Insert Function Call** from the **Code** menu to copy the generated code to the source window.
11. Click on 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 **Sample Oscilloscope** from the **Instrument** menu.
3. Select 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 **Run Function Panel** from the **Code** menu to close the instrument driver.

6. Select **Insert Function Call** from the **Code** menu to copy the generated code to the Source window.
7. Click on the Source window in the background to make it the active window.

Running the Program

Your program source code should match the following text.

```
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 ();

    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.

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"
```

You are now ready to run the program. Select **Run Project** from the **Run** menu.

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 will be 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 **Add File to Project** from the **Edit** menu and choose `instrument (*.fp)` as the file type.
3. Select `scope.fp` from the dialog box.

This concludes this session of the tutorial. In the next chapter you will work some additional exercises to practice what you have learned in these tutorial sessions and learn more about LabWindows/CVI.

Additional Exercises

This is the last chapter of the tutorial sessions. By working the exercises in this chapter, you will learn more about the concepts you have been using throughout this tutorial. Each exercise builds on the code that you developed in the previous exercise. Exercise 1 starts by building on the final sample program that you completed in the Chapter 8 tutorial session. Each exercise gives an outline of the concepts to learn from completing the exercise and some hints for working the exercise.

The solutions to these additional exercises are available in `\cvi\tutorial\solutions`. 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 to continue.

The Base Project

All of the exercises in this chapter build upon the `sample6` project that you completed in the Chapter 8 tutorial session. If you did not complete the previous chapter, go back and do so now. If you have trouble successfully completing the Chapter 8 tutorial, start with the `sample6` project from the `SOLUTIONS` directory.

The `sample6` project generates a waveform and displays it on a graph control when a user clicks on the **Acquire** button. Once you have displayed the data, you can find and display the maximum and minimum values of the datapoints by clicking on 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.

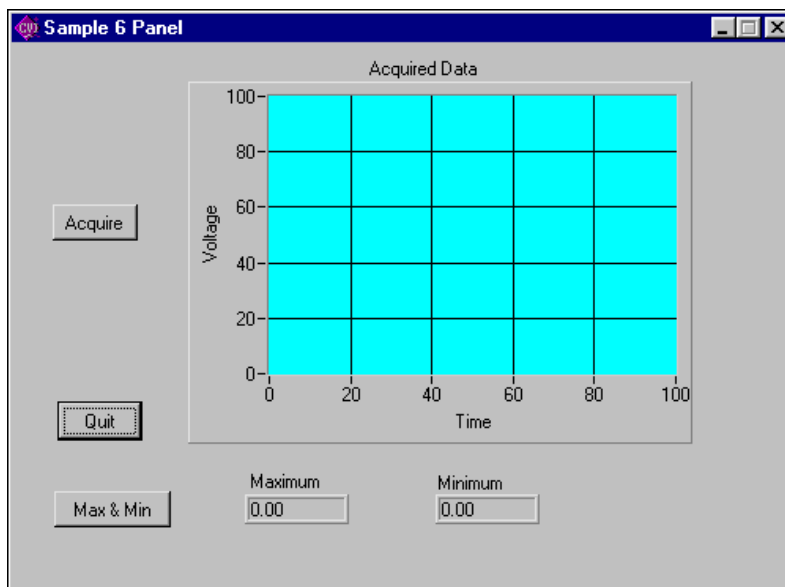


Figure 9-1. Sample User Interface

Exercise 1: Adding a Channel Control

Two of the most common functions you will 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 set the slide control to a specific position or value, you would use `SetCtrlVal`. These functions take the same arguments, a panel handle to determine on which panel the control exists, the control ID to specify which control is to be operated on, and a variable or value that the control will be set to or in which the value of the control will be placed.

Assignment

Because you are using a simulated oscilloscope to *acquire* your data, you may 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.

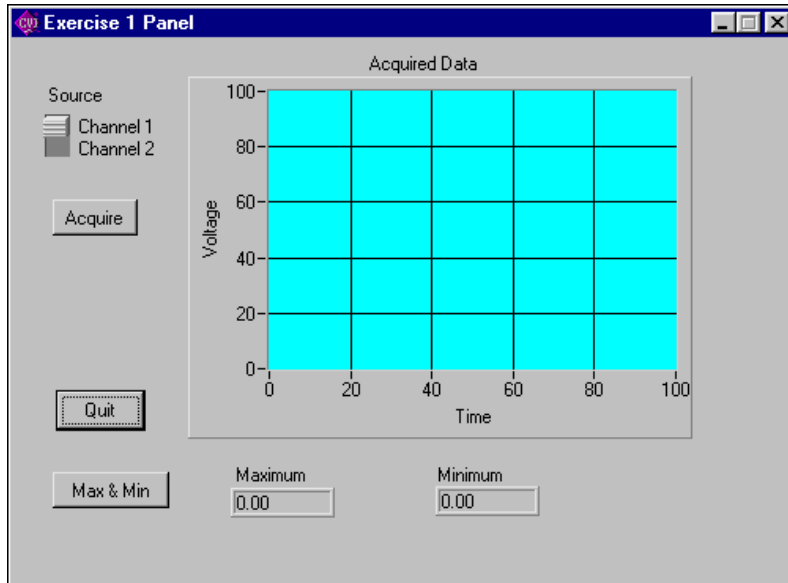


Figure 9-2. User Interface with Channel Selection Control

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 not only build a customized GUI in the User Interface Editor, you can 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 will use the `SetCtrlAttribute` function to change the operation of a command button on the user interface. Because the **Max & Min** command button will not operate correctly until you have acquired the data, it is appropriate to disable the **Max & Min** button until a user clicks on 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 will set is the *dimmed* attribute.

Solution: EXER2.PRJ

Exercise 3: Storing the Waveform to 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 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, 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 will be 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 will be 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 the Chapter 5 tutorial, you used a pop-up panel to display the random number array on a graph (`YGraphPopup`). You can also 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.

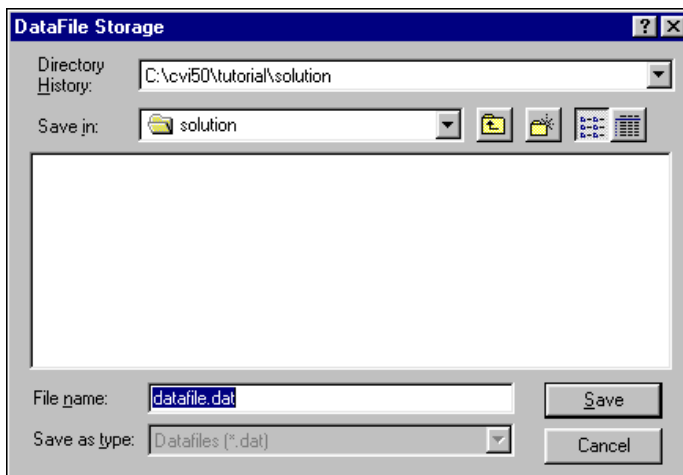


Figure 9-3. File Select Popup

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 file name of the data file. When finished, your `.uir` file should look similar to the one shown in Figure 9-4.

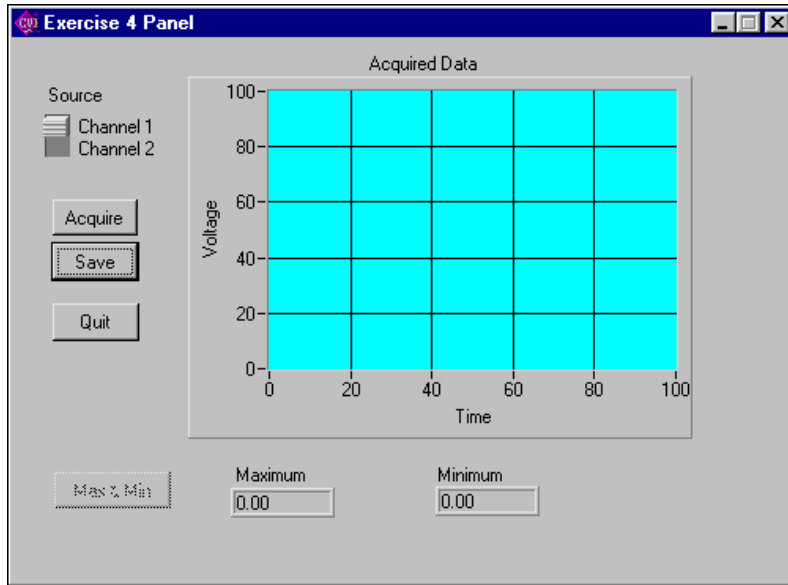


Figure 9-4. Completed User Interface

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 file name 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 only written functions that respond 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. The table below lists all of the events that LabWindows/CVI can generate.

```
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, and 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 only execute when the event is a `COMMIT`. Therefore, the operations in the callback functions only happen 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. Although using a program with a GUI can be intuitive, it might be advantageous to have online help for the controls on the `.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

So far, 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 run-time, the timer control is not displayed. You can specify a Constant Name, Callback function, and timer event interval in the Timer Control edit dialog. LabWindows/CVI will automatically call 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 of the tutorial sessions. The remainder of this manual covers other information that will help you get started using LabWindows/CVI.

Part III

Instrument Control, Data Acquisition, and LabWindows for DOS Conversions

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.

LabWindows/CVI for Windows

- 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)

LabWindows/CVI for Sun

- GPIB-485CT-A (External Serial-GPIB Controller)
- GPIB-SPRC-B (Plug-in SBus interface)
- GPIB-SPRC-S (External SCSI-GPIB Controller)
- GPIB-ENET (External Ethernet TCP/IP-GPIB Controller)

LabWindows/CVI for HP-UX

- GPIB-HP700-EISA (Plug-in card for EISA slots)

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 and NI-488.2M device driver for UNIX 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 **Library Options...** from the **Options** menu in the Project window. Select the GPIB/GPIB 488.2 library option, as shown in Figure 10-1.

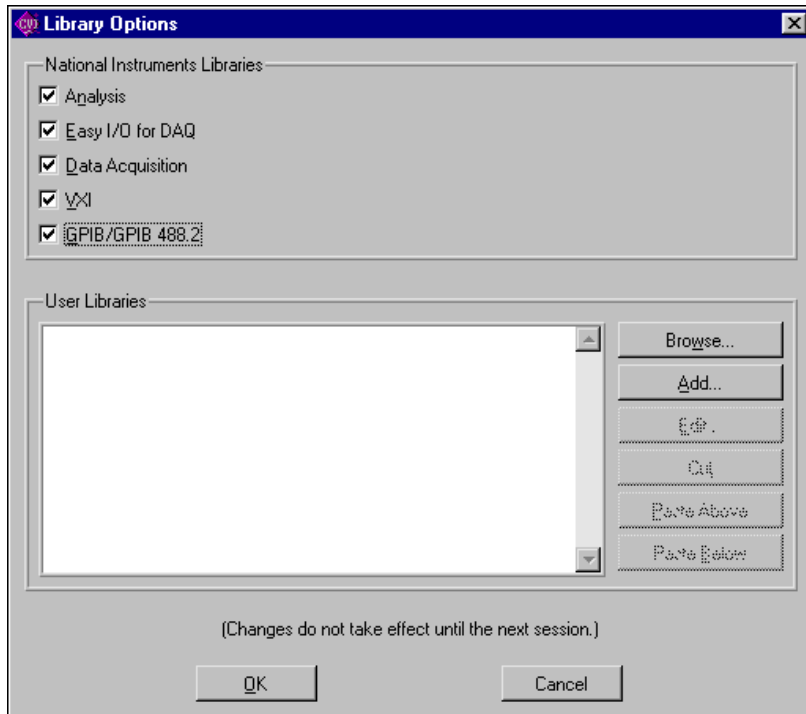


Figure 10-1. Library Options Dialog Box with GPIB/GPIB 488.2 Selected

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. Under Windows, this library is a Dynamic Link Library (DLL). Under UNIX, this library is a device driver. 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 new, fast-growing platform for instrumentation systems. First introduced in 1987, VXI has experienced tremendous growth and acceptance around the world. Today over 50 manufacturers produce more than 650 commercial VXI products. VXI is used in a wide variety of test and measurement and instrument control and 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 boards. Because VXI is based on the VMEbus standard, you can also use VME modules in VXI systems. The VXI widely used 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.

The 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.

LabWindows/CVI for Windows

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

LabWindows/CVI for Sun

VXI-SB 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 for Windows users only. 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 and NI-VXI device driver for UNIX 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 **Library Options** from the **Options** menu in the Project window. Select the VXI library option, as shown in Figure 10-2.

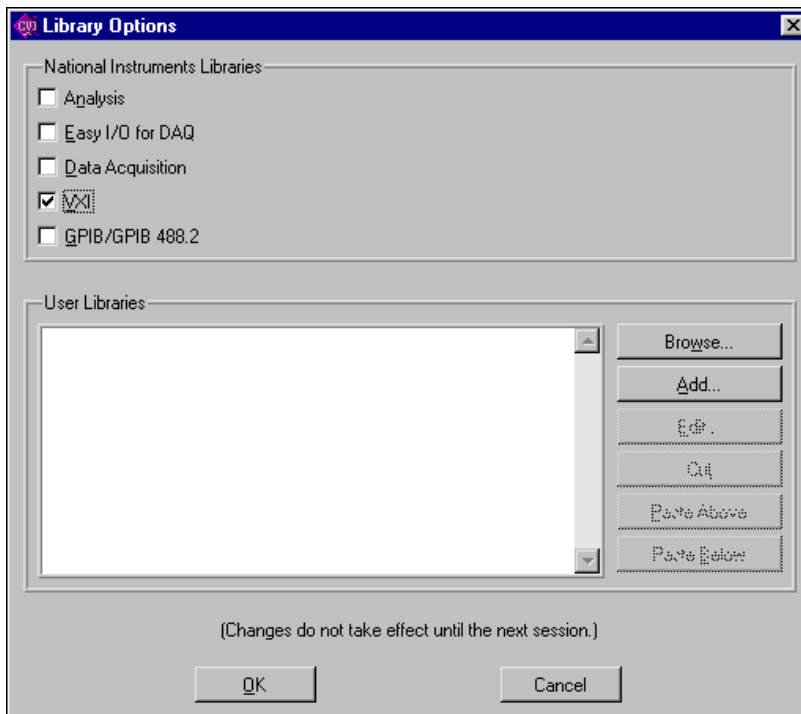


Figure 10-2. Library Options Dialog Box with VXI Selected

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. Under Windows, this library is a DLL. Under UNIX, this library is a shared object. 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 of the low-level command sequences and syntax for your instruments, you can use an instrument driver that builds these command sequences based on inputs from the driver function panels. Therefore, you can communicate with your instrument using intuitive, high-level steps, such as Initialize, Configure, and Measure.

LabWindows/CVI instrument drivers are available to you in source code, so you can optimize the driver to work best for your application. If you have an instrument that is not part of the Instrument Library, you can easily 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*, in this manual. If you plan to develop an instrument driver yourself, refer to the *LabWindows/CVI Instrument Driver Developers Guide* to learn how to use the development tools such as the IVI wizard for creating function trees, function panels, and instrument control source code for a driver.

Getting Started with Data Acquisition

This chapter is a quick reference for installing and configuring National Instruments plug-in data acquisition (DAQ) devices for use with LabWindows/CVI 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.

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 also 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 easily configure your system 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 will occur 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 can also 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, as well as 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 of the files that are required for DAQ operation will be installed.

Configuring LabWindows/CVI for Data Acquisition

Before you can launch LabWindows/CVI and begin programming, you must first configure the software for operation with your DAQ device. Run the NI-DAQ Configuration Utility that was installed by NI-DAQ 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 **Library Options** from the **Options** menu 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.

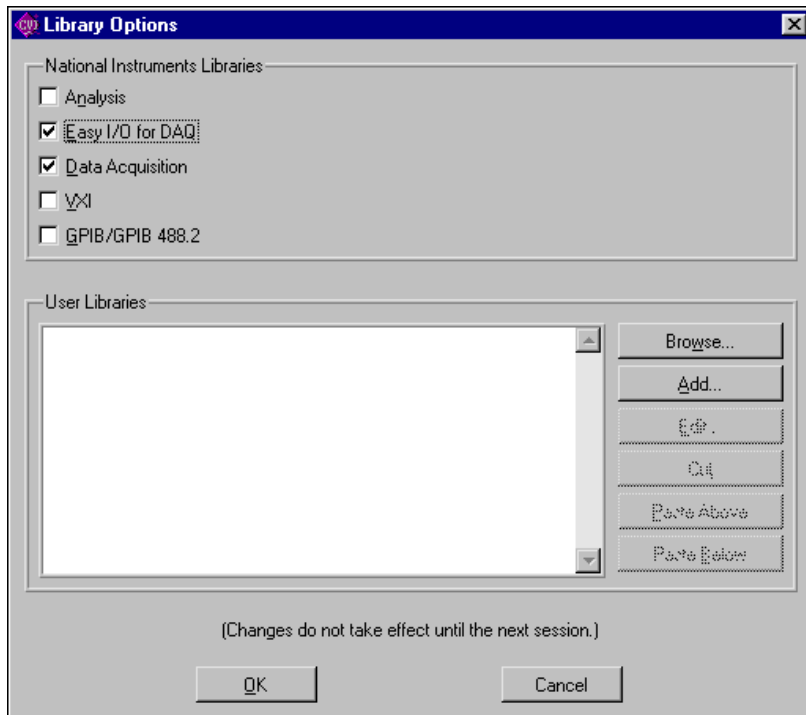


Figure 11-1. Library Options Dialog Box with Easy I/O for DAQ Selected

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 quickly 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 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 **Run Function Panel** from the **Code** menu.
 - AI Sample Channel—The `AI_Sample_Channel` 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 `AO_Sample_Channel` function applies a specified voltage to a single analog output channel.

For more information on how to execute LabWindows/CVI function panels, consult Chapter 5, *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*, of this manual 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, see Chapter 10, *Easy I/O for DAQ Library*, in the *LabWindows/CVI Standard Libraries Reference Manual*.

The sample programs for the Easy I/O for DAQ library are located in the `cvi\samples\easyio` directory. These sample programs are discussed in the EASYIO section of `cvi\samples.doc`.

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\daqnum.fp` when you install LabWindows/CVI. A sample program for this instrument driver is in `cvi\samples\custctrl\daq_num\daqdemo.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\custctrl\daqchart\chartdemo.prj`.

Event Function Parameter Data Types

Some parameters in the Data Acquisition event handling functions that are two bytes under Windows 3.1 have increased in size to four bytes under Windows 95/NT. Typedefs have been added to the include file (`dataacq.h`) and the function panels so that you can write source code that works on all three platforms.

Table 11-1 shows the typedefs and the intrinsic types under for the different platforms.

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

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

The following function prototypes have been 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 have written source code for Windows 3.1 that uses these functions and you want to use the source code under Windows 95/NT, you must modify your source code.

You must change the parameter declarations for all of your event callback functions to match the new callback function prototype. You must also 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 on-line 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*
- *LabWindows/CVI Release Notes*

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

In addition to the sections listed above, 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**—Since 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`, are also 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.
- **RS232 Library**—In LabWindows for DOS, the RS232 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 in the *LabWindows/CVI User Interface Reference Manual* and the *LabWindows/CVI Standard Libraries Reference Manual*.

The 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 containing 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.

The Utility Library

The `GetProgramDir` function has been replaced by the `GetProjectDir` function. The `GetKey` and `KeyHit` functions only work in the Standard Libraries 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 **Open** from the **File** menu from any window. Choose **User Interface (*.uir)** as the file type to open. 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.

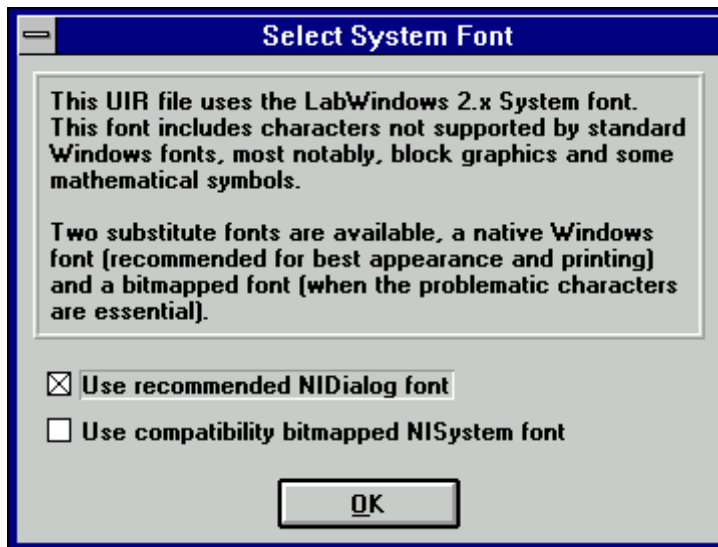


Figure 12-1. Select System Font Dialog Box

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 will 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 both the PC and UNIX platforms.

For the most part, the `.uir` file will appear 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 will run correctly in LabWindows/CVI.

To convert your C source code follow these steps.

1. Open any `.uir` files that your program uses. Refer to the section, [Converting User Interface Resource \(.uir\) Files](#), in this chapter if they have not yet been converted.
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 **Open** from the **File** menu. Choose `Source (*.c)` as the file type to open.
4. Select **Translate LW DOS Program** from the **Options** menu. This opens the dialog box shown in Figure 12-2.

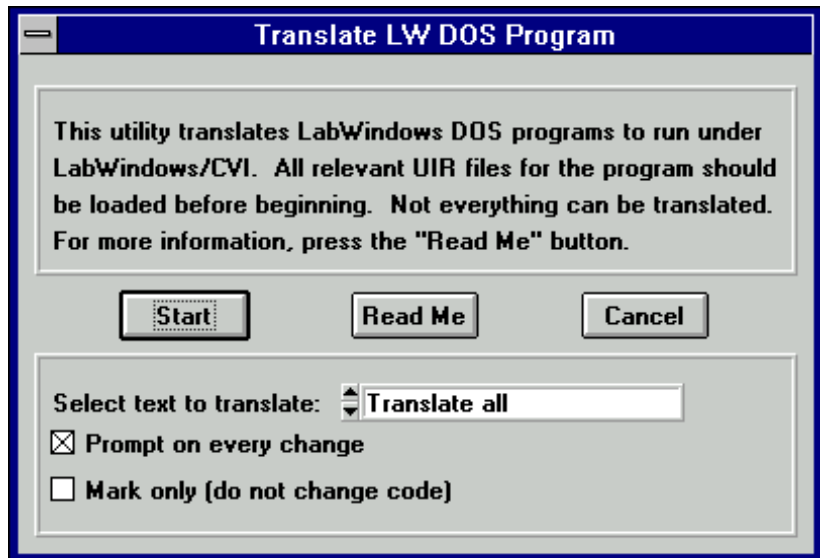


Figure 12-2. Translate LW DOS Program Dialog Box

- **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 have configured the Translate LW DOS Program dialog box, select **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 will need to modify your program so that it checks the return values of the Analysis/Advanced Analysis Library functions for errors.
 7. You will 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. You should declare the array as `short` instead, and 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 will need to modify any `int` variables that you use in a way that requires them to be two-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 will 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 will 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. If your program uses the `[o]` modifier, you will need to modify it as explained in the following paragraphs.

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 both Intel (PC) and Motorola (SPARCstation) architectures, 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 containing 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 two-byte binary data in Intel byte order, your code should appear 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);
```

If, instead, your GPIB instrument sends two-byte binary data in Motorola byte order, the `Scan` function should appear as follows.

```
Scan (instr_buf, "%100d[b2o10]>%100d", prog_buf);
```

In either case, the `o` modifier is used only on the buffer containing the raw data from the instrument (`instr_buf`). LabWindows/CVI will ensure that the program buffer (`prog_buf`) is in the proper byte order for the host processor. For a full description of the `o` modifier, see Chapter 2, *Formatting and I/O Library*, in the *LabWindows/CVI Standard Libraries Reference Manual*.

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 should acquire new LabWindows/CVI instrument drivers from the National Instruments web site or ftp site. Refer to Appendix A, Customer Communication, 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. The steps are as follows.

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

Convert the Instrument Driver Function Panels

1. Double-click on 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 Function Panel Window from the Edit menu.
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 Transfer Window Help to Function Help from the Options menu.
6. Save the *.fp file.

Convert the Instrument Driver Header File

1. Load the *.h file by double-clicking 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. Load the *.c file by double clicking on it in the Project window.
2. Continue with step 3 in the section, [Converting Source Code](#), 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 section, *Converting Source Code*, 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 Programmers 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. Under Windows, these modules will have to be rewritten using the Microsoft DDK. Under UNIX, you will have to rewrite hardware-specific modules using the host system library.



Customer Communication

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

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

Electronic Services

Bulletin Board Support

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

United States: 512 794 5422

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

United Kingdom: 01635 551422

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

France: 01 48 65 15 59

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

FTP Support

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

Fax-on-Demand Support

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

E-Mail Support (Currently USA Only)

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

support@natinst.com

Telephone and Fax Support

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

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

Technical Support Form

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

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

Name _____

Company _____

Address _____

Fax (____) _____ Phone (____) _____

Computer brand _____ Model _____ Processor _____

Operating system (include version number) _____

Clock speed _____ MHz RAM _____ MB Display adapter _____

Mouse ___ yes ___ no Other adapters installed _____

Hard disk capacity _____ MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

List any error messages: _____

The following steps reproduce the problem: _____

LabWindows/CVI Hardware and Software Configuration Form

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

National Instruments Products

DAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

NI-DAQ, LabVIEW, or LabWindows version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

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

Title: *Getting Started with LabWindows/CVI*

Edition Date: February 1998

Part Number: 320680D-01

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

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

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

E-Mail Address _____

Phone (____) _____ Fax (____) _____

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

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
μ -	micro-	10^{-6}
n-	nano-	10^{-9}

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

check box 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.

E

entry mode indicator	An element of the LabWindows/CVI screen that indicates the current text entry mode as either insert or overwrite.
----------------------	---

F

.fcp file	A file that contains information about the function tree and function panels of an instrument driver.
full-screen mode	A screen display mode in which one window occupies the entire screen.

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

in.	inches
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 may be executed without creating an entire program.

L

- `.lbw` file A file that contains code stored in LabWindows for DOS binary format, a format translatable into C or BASIC source code by the LabWindows for DOS environment.
- list box A dialog box item that displays a list of possible choices for completing a command in the dialog box.
- `.lwi` file A file that contains instrument include statements in LabWindows for DOS binary format.

M

- MB megabytes of memory
- menu An area accessible from the menu 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 upon 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 drop-and-drag 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, as well as 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, 8-13
 - constructing projects, 6-11
 - illustration, 2-6
- AISampleChannel function panel, 11-4
- ANSI C Library, 1-4
- ANSI C specifications, 2-9
- AOSampleChannel function, 11-4
- Array Display window
 - displaying arrays, 4-11 to 4-12
 - editing arrays, 4-12
- arrays
 - declaring from function panels, 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, 4-3
- 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 4-6

- Build Error window, 2-7
- Build Errors command, Window menu, 2-7
- Build menu, Project window, 2-5
- bulletin board support, A-1

C

- callback functions
 - adding to Save button, 9-7
 - adding with CodeBuilder, 7-3 to 7-4
 - 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 selection control, adding to project, 9-2 to 9-3
- Chart Control Instrument Driver, DAQ, 11-5
- Close command, File menu, 2-10
- 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 UI Constant command, 6-8
 - Set Target File command, 3-7, 6-9
 - User Interface Editor, 5-12

- CodeBuilder
 - adding control callback function, 7-3 to 7-4
 - generating program shell, 1-6, 5-11 to 5-14
 - locating FindMaxMin callback function, 7-6 to 7-7
 - Command button
 - adding for numeric control, 7-3 to 7-6
 - 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, 2-7
 - conditional breakpoints, 4-4
 - configuration
 - data acquisition boards, 11-2
 - testing, 11-4
 - GPIB driver software, 10-2
 - instrument drivers, 8-5 to 8-7
 - NI-DAQ driver software, 11-3
 - VXI driver software, 10-5
 - VXI hardware, 10-5
 - constants, displaying in .uir file, 6-8 to 6-9
 - Continue command, Run menu, 4-3
 - Continue Execution command, Run menu, 4-14
 - Control Callback Events dialog box, 5-11
 - 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, 12-8
 - header file, 12-8
 - source code, 12-8
 - loadable compiled modules or external modules, 12-9
 - source code, 12-5 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
 - customer communication, xvii, A-1 to A-2
 - customizing toolbars, 2-9
- ## 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, 11-3
- testing operation and configuration, 11-4
- Data Acquisition Library
 - LabWindows for DOS, 12-2
 - sample programs, 11-5
- data analysis
 - generating a call to Mean function, 3-9 to 3-10
 - programming overview, 1-7
- data files, functions for reading and writing, 9-5
- data types for event functions, changes in, 11-6 to 11-7
- debugging programs
 - breakpoints, 4-4 to 4-6
 - invoking, 4-4
 - manual breakpoints, 4-6
 - programmatically breakpoints, 4-4 to 4-6
 - 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, 4-3
- Declare Variable command, Code menu
 - declaring arrays (example), 8-9
 - declaring variables (example), 7-7
 - reading waveform (example), 8-10
- developing graphical user interfaces (GUI). *See* graphical user interface (GUI), building.
- direct memory access (DMA) channel for data acquisition boards, 11-2
- displaying and editing data
 - 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, *xv-xvi*
 - data acquisition boards, 11-7
 - manuals for VXI controller boards, 10-5
 - optional manuals, *xvii*
 - organization of manual, *xiii-xv*
 - standard LabWindows/CVI documentation set, *xvi*

E

- Easy I/O for DAQ Library
 - sample programs, 11-5
- Edit menu
 - Project window
 - Add File to Project command, 2-6, 6-11, 8-13
 - overview, 2-5
 - Source, Interactive Execution, and Standard Input/Output windows editing tools, 2-9 to 2-11
 - Undo command, 2-10
- editing data
 - arrays, 4-12
 - strings, 4-13
 - variables, 4-9 to 4-11
- editing tools in Source window, 2-9 to 2-11
- electronic support services, A-1 to A-2
- e-mail support, A-2
- error messages
 - during compiling and linking, 2-7
 - removing from screen, 2-7
- Error window. *See also* Build Error window.
 - closing and opening, 2-7
- EVENT_COMMIT, 7-9, 9-8
- EVENT_GOT_FOCUS, 7-9, 9-8
- EVENT_LEFT_CLICK, 7-9, 9-8
- EVENT_RIGHT_CLICK, 5-11

- events
 - event function parameter data type changes, 11-6 to 11-7
 - Windows 3.1 source code changes required, 11-7
 - timed events, 9-9
 - 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, 12-9
- F**
- fax and telephone support numbers, A-2
 - Fax-on-Demand support, A-2
 - file I/O functions, ANSI C library, 9-5
 - File menu
 - Project window
 - Close command, 2-10
 - Open command, 2-3
 - overview, 2-5
 - Source, Interactive Execution, and Standard Input/Output windows
 - Hide command, 2-8
 - Open Quoted Text command, 2-9
 - User Interface Editor, 5-4
 - File Select Popup, 9-6
 - Find Function Panel command,
 - View menu, 3-9
 - finding functions, 3-9 to 3-10
 - 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
 - FTP support, A-1
 - Function command, Help menu, 3-5, 3-6
 - function panel controls
 - generating events, 9-8 to 9-9
 - input, 3-5
 - output, 3-10 to 3-11
 - function panels
 - adding YGraphPopup function to
 - sample program, 3-7 to 3-8
 - analyzing data, 3-9 to 3-10
 - controls, 3-5
 - converting instrument driver
 - function panels, 12-8
 - declaring arrays, 8-9
 - definition, 3-5
 - drawing a graph, 3-6 to 3-7
 - executing functions interactively,
 - 8-3 to 8-4
 - finding functions, 3-9 to 3-10
 - function definitions, 4-3
 - PlotY function in User Interface Library, 6-5 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, 3-10 to 3-11
 - purpose and use, 3-5
 - recalling, 3-11 to 3-12
 - selecting, 6-5 to 6-7
 - testing operation of DAQ device and configuration, 11-4
 - function trees, 3-3

G

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

Generate All Code command,
Code menu, 5-12

Generate All Code dialog box, 5-12

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 toolbar icon, 4-3

GPIB boards
compatible LabWindows/CVI
interfaces, 10-1
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, 6-5 to 6-7

graphical user interface (GUI)
building. *See* graphical user interface (GUI), building.
components (figure), 1-5
definition, 1-6
example of GUI, 2-11 to 2-12
illustration, 2-12

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, 9-7

building PlotY function call syntax,
6-8 to 6-10

building user interface resource (.uir) file,
5-3 to 5-14

constructing the project, 6-10 to 6-11

finding PlotY function, 6-5 to 6-7

main function, 6-2

modifying, 7-2 to 7-6

running the program, 6-12, 7-9

saving .uir files, 5-9 to 5-10

setting attributes programmatically, 9-4

Shutdown function, 6-3

source code requirements, 5-3

using instrument driver, 8-3 to 8-11

waveform generation project
(illustration), 9-2
writing callback function, 7-6 to 7-8

Graphics Library (LabWindows for
DOS), 12-2

graphs, drawing with function panels,
3-6 to 3-7

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

H

header files
for instrument drivers, converting from
LabWindows for DOS, 12-8
for user interface, viewing, 5-10

help information, adding to command
buttons, 9-9

Help menu, Function Panel windows, 3-5, 3-6

Hide command, File menu, 2-8

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, 8-11
 - inserting code from function panel, 3-7 to 3-8
 - PlotY function panel example, 6-9
 - installation
 - data acquisition boards, 11-2
 - general instructions, 1-1 to 1-2
 - GPIO boards, 10-2
 - LabWindows/CVI subdirectories (table), 1-1 to 1-2
 - NI-DAQ driver software, 11-2
 - VXI controllers, 10-5
 - instrument driver programming example, 8-1 to 8-13
 - adding sample program to project files, 8-13
 - appearance of Sample Oscilloscope program on Instruments menu, 8-2
 - closing instrument drivers, 8-10 to 8-11
 - configuring instrument drivers, 8-5 to 8-7
 - declaring arrays from function panels, 8-9
 - functions in Sample Oscilloscope instrument, 8-3
 - initializing instrument drivers, 8-4 to 8-5
 - interactive function panel execution, 8-3 to 8-4
 - loading, 8-1 to 8-2
 - reading data, 8-7 to 8-8
 - reading waveform, 8-10
 - running the sample program, 8-11 to 8-13
 - instrument drivers
 - available drivers, 10-7
 - converting from LabWindows for DOS, 12-7 to 12-8
 - function panels, 12-8
 - 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, 1-4
 - Instrument menu, 8-2
 - interactive code generation tools. *See also* CodeBuilder.
 - accessing User Interface Library, 3-3 to 3-4
 - analyzing data, 3-9 to 3-10
 - drawing graphs using function panels, 3-6 to 3-7
 - executing function panels interactively, 3-14
 - finishing the sample program, 3-13
 - 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, 3-2 to 3-3
 - output values on function panels, 3-10 to 3-11
 - recalling function panels, 3-11 to 3-12
 - 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.
 definition, 1-2
 features, 1-3 to 1-4
 learning to use, 1-2 to 1-3
 setting up the tutorial, 2-2 to 2-9
 starting, 2-2
 VXI development system (VXS), 10-4

Library menu
 function tree (illustration), 3-3
 selecting libraries, 3-2
 User Interface command, 6-5

Library Options command, Options menu
 loading data acquisition libraries, 11-3
 loading GPIB libraries, 10-3
 loading VXI libraries, 10-6

Line command, View menu, 2-11

Line Numbers command, View menu, 2-9

linker error messages, 2-7

Load button, 3-1

loadable compiled modules, converting from
 LabWindows for DOS, 12-9

loading projects, 2-3 to 2-5

LoadPanel function, 6-2

M

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, 3-9 to 3-10

MessagePopup function, 9-9

Minimum Index control, 7-7

Minimum Value control, 7-7

N

National Instruments libraries, 1-4. *See also*
 standard libraries; specific libraries.

Next Tag command, View menu, 2-11

NI-488.2 DLL device driver, 10-3

NI-488.2 Library, 10-3

NI-488.2M DLL device driver, 10-3

NI-DAQ driver software
 configuration
 for data acquisition, 11-3
 NI-DAQ Configuration Utility, 11-3
 installation, 11-2

NI-Spy utility, 10-5

NI-VXI library, 10-6

Numeric command, Create menu, 7-5

numeric controls
 adding to user interface, 7-4 to 7-5
 DAQ Numeric Control
 Instrument Driver, 11-5

O

Open command, File menu, 2-3

Open Quoted Text command, File menu, 2-9

Open User Interface command, File menu, 5-4

Options menu, Project window
 Library Options command,
 10-3, 10-6, 11-3
 overview, 2-5
 Toolbar command, 2-9
 Translate LW DOS Program
 command, 12-5

Oscilloscope, sample. *See* instrument driver
 programming example.

output controls, adding to function panel,
 3-10 to 3-11

P

PlotY function

- building call syntax, 6-8 to 6-10

- finding in User Interface Library, 6-5 to 6-7

- function panel (illustration), 6-9

pop-up panels, 9-6 to 9-7

- adding to Command buttons, 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, 5-10

program control

- components (figure), 1-5

- overview, 1-6

program development overview, 1-4 to 1-7.

- See also* source files.

- program structure for LabWindows/CVI, 1-5 to 1-7

- data acquisition, 1-6 to 1-7

- data analysis, 1-7

- program control, 1-6

- program shell generation with CodeBuilder, 1-6

- relationship between structures (figure), 1-5

- user interface, 1-6

- using C in LabWindows/CVI, 1-5

program shell, building. *See* CodeBuilder.

programmatically breakpoints

- invoking, 4-4

- performing, 4-4 to 4-6

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.

debugging programs

- breakpoints, 4-4 to 4-6

- manual breakpoints, 4-6

- programmatically breakpoints, 4-4 to 4-6

- 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, 4-3

- editing tools, 2-9 to 2-11

function panels

- adding YGraphPopup function to sample program, 3-7 to 3-8

- analyzing data, 3-9 to 3-10

- controls, 3-5

- declaring arrays, 8-9

- drawing a graph, 3-6 to 3-7

- executing functions interactively, 8-3 to 8-4

- finding functions, 3-9 to 3-10

- finding PlotY function in

- User Interface Library, 6-5 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, 3-10 to 3-11

- recalling, 3-11 to 3-12

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, 9-7

- 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 5-14
- constructing the project, 6-10 to 6-11
- definition, 1-6
- example of GUI, 2-11 to 2-12
- finding PlotY function, 6-5 to 6-7
- generating random array of data, 6-3 to 6-4
- main function, 6-2
- modifying, 7-2 to 7-6
- running the program, 6-12, 7-9
- saving .uir files, 5-9 to 5-10
- setting attributes
 - programmatically, 9-4
- Shutdown function, 6-3
- source code requirements, 5-3
- waveform generation project (illustration), 9-2
- writing callback function, 7-6 to 7-8

instrument driver example, 8-1 to 8-13

- adding sample program to project files, 8-13
- closing, 8-10 to 8-11
- configuring, 8-5 to 8-7
- declaring arrays from function panels, 8-9
- functions in Sample Oscilloscope instrument, 8-3
- initializing, 8-4 to 8-5
- interactive function panel execution, 8-3 to 8-4
- interactive use, 8-3 to 8-4
- loading, 8-1 to 8-2
- reading data, 8-7 to 8-8
- reading waveform, 8-10
- running the sample program, 8-11 to 8-13

- Library menu, 3-2 to 3-3
- loading projects, 2-3 to 2-5
- running projects, 2-7
- windows used for programming, 2-1 to 2-2

Project window

- adding files to projects, 2-6 to 2-7
- Build menu, 2-5
- Edit menu, 2-5
- File menu, 2-5
- illustration, 2-1
- information displayed about current project (figure), 2-7
- loading projects, 2-3 to 2-5
- Options menu, 2-5
- Run menu, 2-5, 2-7
- title determined by current project, 2-1
- Transfer Project Options dialog box (illustration), 5-2
- Window menu, 2-5

projects. *See also* source files.

- adding files to projects, 2-6 to 2-7, 6-11, 8-13
- adding instruments, 8-13
- error messages, 2-7
- loading, 2-3 to 2-5
- running
 - instrument driver example, 8-11 to 8-13
 - Max & Min command button sample, 7-9
 - sample1 project, 2-7
 - sample4.prj, 6-12

R

- random array of data, generating, 6-3 to 6-4
- Read Waveform function example, 8-10
- reading data with instrument driver, 8-7 to 8-8
- Recall Function Panel command, View menu, 3-12

RESMAN utility, 10-5
 right-clicking on GUI control, 5-11
 RS232 Library (LabWindows for DOS), 12-2
 Run Function Panel command, Code menu, 8-5, 8-7, 8-10
 Run menu
 Interactive Execution window
 Continue command, 4-3
 Continue Execution command, 4-14
 Step Into command, 4-3, 4-6
 Step Over command, 4-3
 Terminate Execution command, 4-3
 Project window
 Break at First Statement command, 4-3
 overview, 2-5
 Run Project command, 2-7, 3-13, 4-3, 6-12
 Run Project command, Run menu
 running completed project, 3-13, 6-12
 sample1 project, 2-7
 step mode execution, 4-3
 running projects
 instrument driver example, 8-11 to 8-13
 Max & Min command button sample, 7-9
 sample1 project, 2-7
 sample4.prj, 6-12
 Runtime Errors command, Window menu, 2-7
 RunUserInterface function, 6-2

S

Sample Oscilloscope program. *See* instrument driver programming example.
 sample programs. *See* example programs.
 Save button, adding to project, 9-7
 scope.fp sample file, 8-1
 Select UI 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
 SOLUTIONS 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, 12-5 to 12-7
 instrument drivers, 12-8
 displaying in Source window, 2-8 to 2-9
 displaying referenced files, 2-9
 error messages, 2-7
 information displayed in Project window (figure), 2-7
 inserting code from function panels, 3-7 to 3-8
 loading, 2-3 to 2-5
 moving to specific lines of code, 2-11
 opening subwindows for one source file, 2-10
 recalling function panel for editing, 3-11 to 3-12
 Source window
 available menus, 2-9
 compatibility with ANSI C specifications, 2-9
 displaying generated code, 5-13
 displaying source code, 2-8 to 2-9
 editing tools, 2-9 to 2-11
 illustration, 2-2, 2-8
 opening subwindows, 2-10
 sample3.c program (illustration), 4-2
 title determined by current source file, 2-1

Standard Input/Output window, 2-8
 standard libraries, 1-4
 starting LabWindows/CVI, 2-2
 Step Into command, Run menu, 4-3, 4-6
 step mode execution, 4-3
 Step Over command, Run menu, 4-3
 String Display window
 displaying and editing
 string variables, 4-13
 illustration, 4-13
 subwindows
 illustration, 2-10
 opening subwindows for
 one source file, 2-10
 Sun systems, differences in
 LabWindows/CVI documentation, 1-2

T

T&M Explorer utility, 10-5
 tagged lines, 2-11
 technical support, A-1 to A-2
 telephone and fax support numbers, A-2
 Terminate Execution command,
 Run menu, 4-3
 testing board configuration, 11-4
 using function panels, 11-4
 timed events, 9-9
 timer controls, 9-9
 Toggle Tag command, View menu, 2-11
 Toolbar command, Options menu, 2-9
 toolbars
 customizing, 2-9
 displaying names of button or icons, 2-9
 Transfer Project Options dialog box
 (illustration), 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, 2-10
 user interface. *See* graphical
 user interface (GUI).
 user interface, creating. *See* graphical
 user interface (GUI), building.
 User Interface command, Library menu, 6-5
 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, 1-4
 finding PlotY function, 6-5 to 6-7
 functions with new behavior, 12-3
 illustration, 3-4
 Pop-up Panels, 3-4, 9-6 to 9-7
 support for LabWindows for
 DOS features, 12-2
 user interface resource (.uir) files
 building, 5-3 to 5-14
 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 to 5-10
 Utility Library
 functions with new behavior, 12-3
 support for LabWindows for DOS
 features, 12-2

V

- variables
 - displaying, 4-7 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, 3-9
 - Preview User Interface Header File, 5-10
 - Recall Function Panel command, 3-12
 - Source, Interactive Execution, and Standard Input/Output windows
 - Line command, 2-11
 - Next Tag command, 2-11
 - Toggle Tag command, 2-11
- VME eXtensions for Instrumentation (VXI).
 - See* VXI controllers.
- VXI controllers
 - compatible controllers, 10-5
 - configuration
 - configuring LabWindows/CVI for VXI, 10-6
 - VXI driver software, 10-5
 - VXI hardware, 10-5
 - developing applications, 10-6
 - documentation for VXI controller boards, 10-5
 - installation, 10-5
 - LabWindows/CVI VXI development system (VXS), 10-4
 - overview, 10-4

W

- Watch window
 - 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 selection control, 9-2 to 9-3
 - Read Waveform sample function, 8-10
 - storing waveform to disk, 9-5
 - user interface (illustration), 9-2
- Window menu, Project window
 - Build Errors command, 2-7
 - overview, 2-5
 - Runtime Errors command, 2-7
- windows
 - managing windows (note), 2-2
 - program development windows, 2-1 to 2-2
- Windows 95/NT, event function parameter data type changes, 11-6 to 11-7
 - source code changes required, 11-7

Y

- Y Array control, 3-6
- YGraphPopup function, 3-7 to 3-8