Getting Started With
LabWindows®/CVI

July 1996 Edition

Part Number 320680C-01

Internet Support

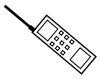
GPIB: gpib.support@natinst.com
DAQ: daq.support@natinst.com
VXI: vxi.support@natinst.com
LabVIEW: lv.support@natinst.com
LabWindows: lw.support@natinst.com
HiQ: hiq.support@natinst.com
VISA: visa.support@natinst.com
FTP Site: ftp.natinst.com
Web Address: www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422 or (800) 327-3077
BBS United Kingdom: 01635 551422
BBS France: 1 48 65 15 59

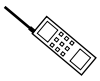
FaxBack Support

(512) 418-1111



Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678



International Offices

Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico 95 800 010 0793,
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, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Warranty

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

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

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

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

Copyright

Trademarks

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

Part III: Tutorial—Instrument Control, Data Acquisition, and Labwindows for DOS Conversions

About This Manual

Getting Started with LabWindows/CVI is a hands-on introduction to the LabWindows/CVI software package. This manual is intended for use by 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 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 Variable Display, 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 being generated. 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. You are given an outline of the concepts to learn from completing the exercises and some hints for working the exercises.

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) boards for use with LabWindows/CVI for DOS. This chapter discusses how to install and configure both hardware and software, and how to test the board operation.
- 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:

bold	Bold text denotes a parameter, menu item, return value, function panel item, or dialog box button or option.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
bold italic	Bold italic text denotes a note, caution, or warning.
`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 program code.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
< >	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.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in `drivename\dir1name\dir2name\myfile`

Acronyms, abbreviations, metric prefixes, mnemonics, and symbols, and terms are listed in the Glossary.

The LabWindows/CVI Documentation Set

For a detailed discussion of the best way to use the LabWindows/CVI documentation set, see the section *Using the LabWindows/CVI Documentation Set* in Chapter 1, *Introduction to LabWindows/CVI* of this manual.

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 Function Reference Manual for DOS/Windows* and the *NI-488.2M Software Reference Manual* describes a library of functions you can use to program National Instruments GPIB interfaces. These manuals are 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 Software Reference Manual for C* describes 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, *Customer Communication*, at the end of this manual.

Chapter 1
Introduction to LabWindows/CVI

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

Installing LabWindows/CVI

The 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.

LabWindows/CVI and the associated files are installed in the following subdirectories on your hard disk.

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
`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 (LabWindows/CVI Full Development System 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

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

Note to Sun Users

The figures used throughout the LabWindows/CVI documentation set are generated 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 (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 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 those volumes. However, it is still useful to make quick cross references to other manuals as questions arise, both while learning and while using LabWindows/CVI. When you begin the tutorial in Chapter 2 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 may have. In addition, descriptions of these windows and their menus are in LabWindows/CVI online help.

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 as well as 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* complete with online help that lets you interactively execute the function and generate code for calling the function.

Programs written 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. See 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, there are seven libraries—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, there are three libraries—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, there are three libraries—the Dynamic Data Exchange (DDE) Library (Windows version only), the Transmission Control Protocol (TCP) Library, and the X Property Library.

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, please review Chapter 1, *The ANSI C Library*, in the *LabWindows/CVI Standard Libraries Reference Manual*.

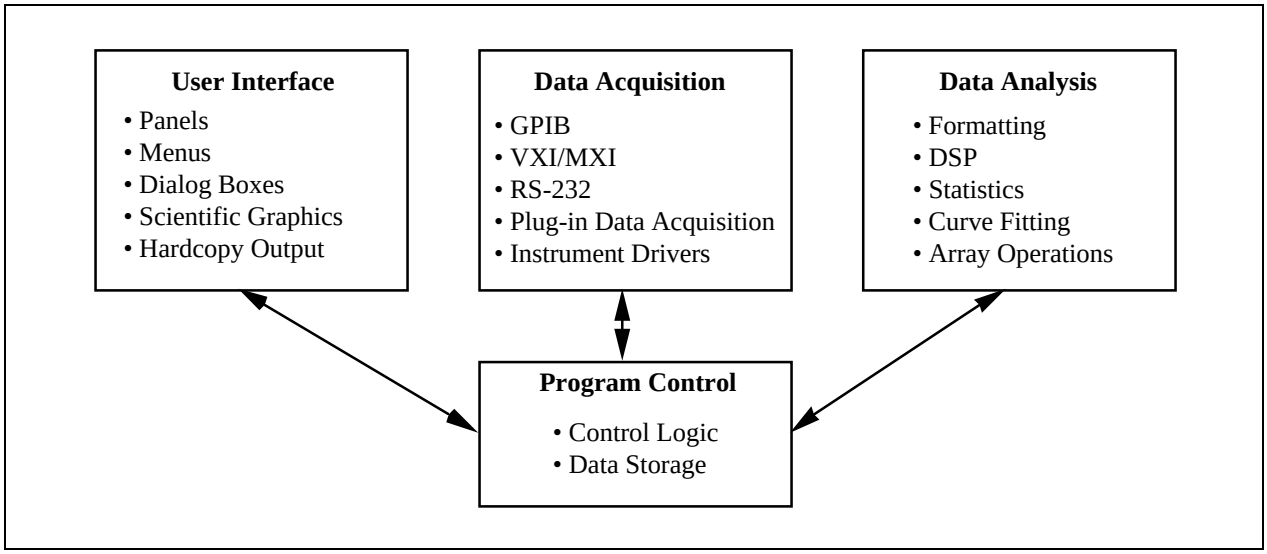
It is also helpful to examine the sample program listings distributed with LabWindows/CVI. These examples illustrate both the LabWindows/CVI User Interface features and the manner in which you can use the LabWindows/CVI library functions.

LabWindows/CVI Program Structure

Once 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

These program elements are interrelated as shown in the following illustration.



While this illustration 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

Once 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. There are functions 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 software reference manual that comes with your GPIB interface. VXI library functions are documented in the *NI-VXI Software Reference Manual for C* 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 *Instruments Menu* sections of Chapter 3, *The Project Menu* in the *LabWindows/CVI User Manual*.

Additionally, Chapter 10, G*etting 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 acquiring 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.

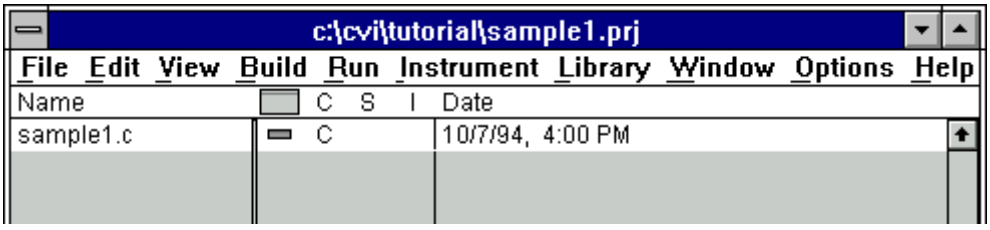
Chapter 2
Loading, Running, and Editing Source Code

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.

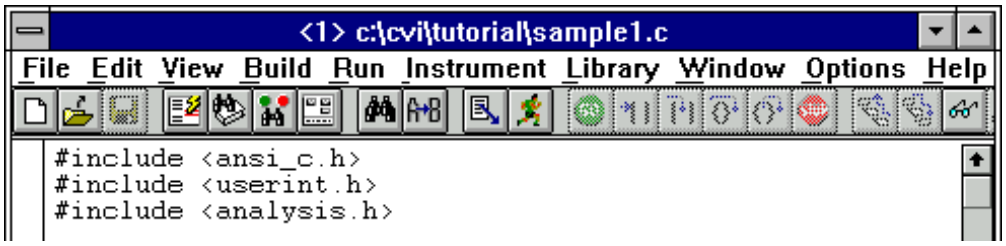
- Some of the windows in LabWindows/CVI.
- How to load and operate projects in LabWindows/CVI.
- The different types of files that can be used in a LabWindows/CVI project.
- Some of the source code editing techniques available in LabWindows/CVI.

A Note about LabWindows/CVI Windows

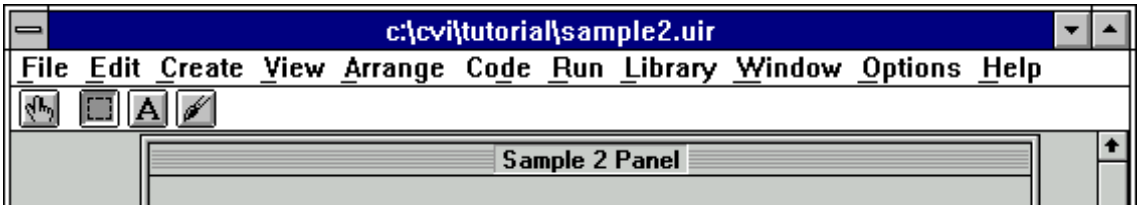
The three main windows you will be using in this tutorial are the Project window, the Source window, and the User Interface Editor window. Unlike other windows in the LabWindows/CVI program, these three windows are titled according to the program you are working on. The file extension will be your clue 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 are shown in the following illustrations.



The Project Window



The Source Window



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 is launched, you see an empty Project window.

Note: ***All windows in LabWindows/CVI can be manipulated 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 Microsoft 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.

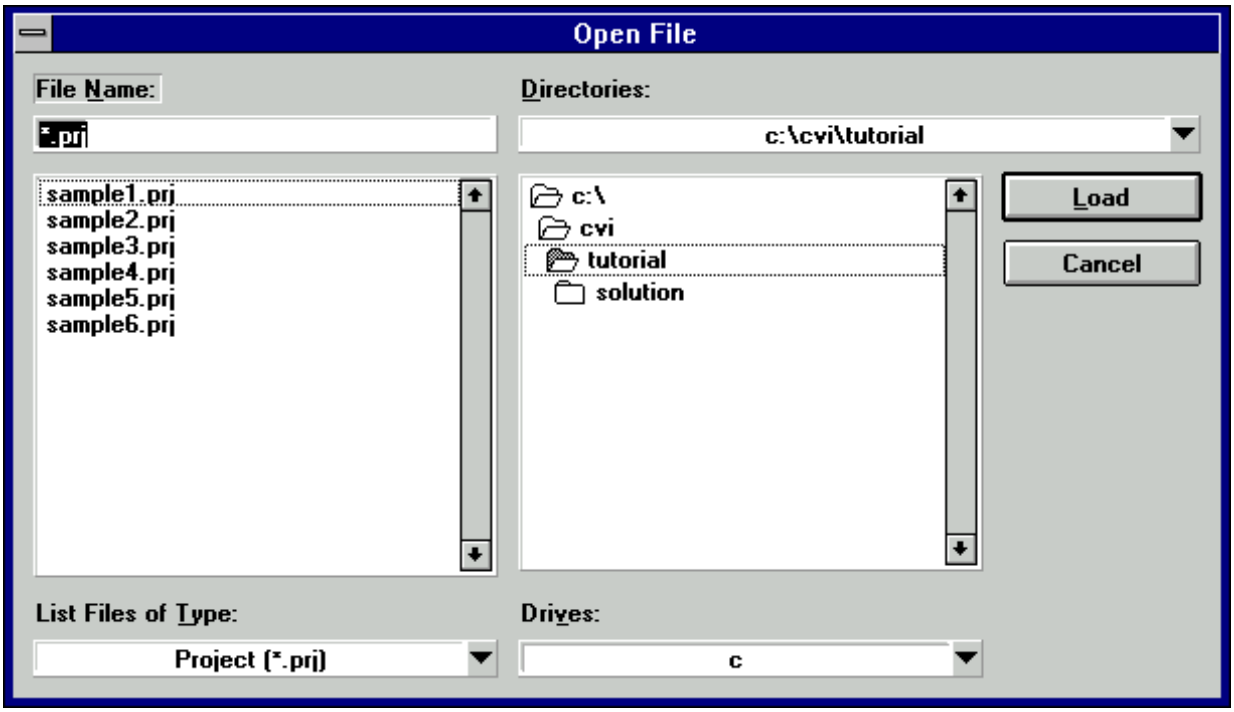
If other people will be using the Getting Started example files on your computer, be sure to use the Save As... option to save your files under different file names.

Loading a Project into LabWindows/CVI

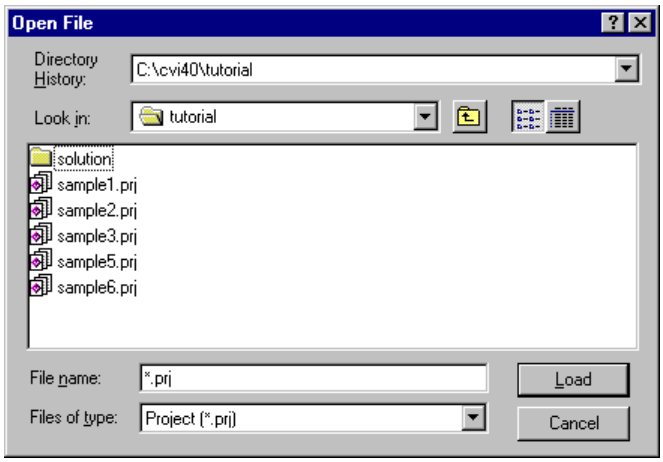
To view some of the editing and execution features of the LabWindows/CVI development, you will load a project into the LabWindows/CVI Project window. Follow these instructions.

1. Select **Open** from the **File** menu. You will be presented with a list of the different file types that can be created and edited in LabWindows/CVI.

2. Select `Project (*.prj)` to bring up the Open File dialog box shown in the following illustration. You may have to click on the `CVI` directory, then click on the `tutorial` directory to bring up the project files.



Open File Dialog Box in Windows 3.1

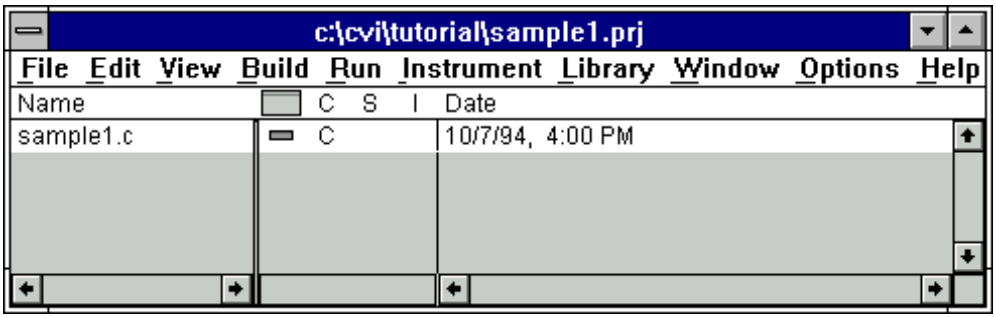


Open File Dialog Box in Windows 95

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

After loading the `sample1.prj` project, the Project window should appear as shown in the following illustration.

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



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

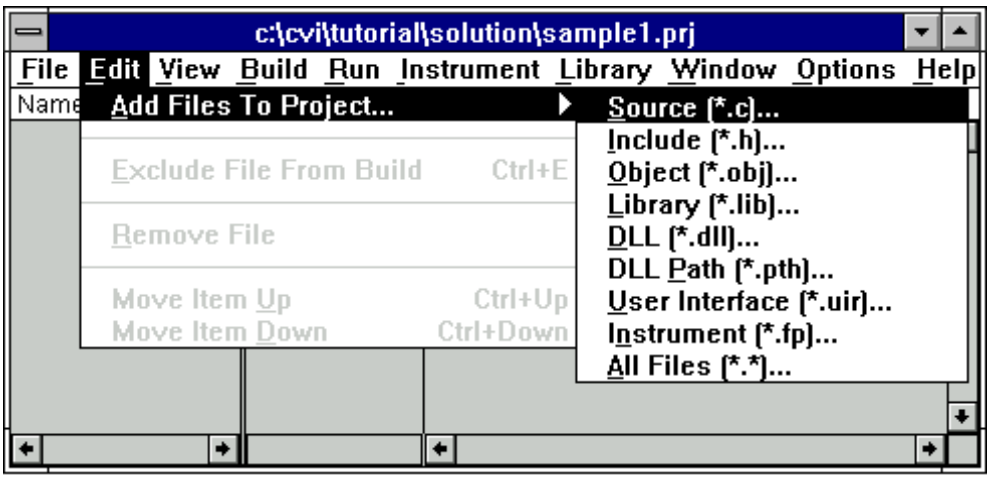
The Project Window

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

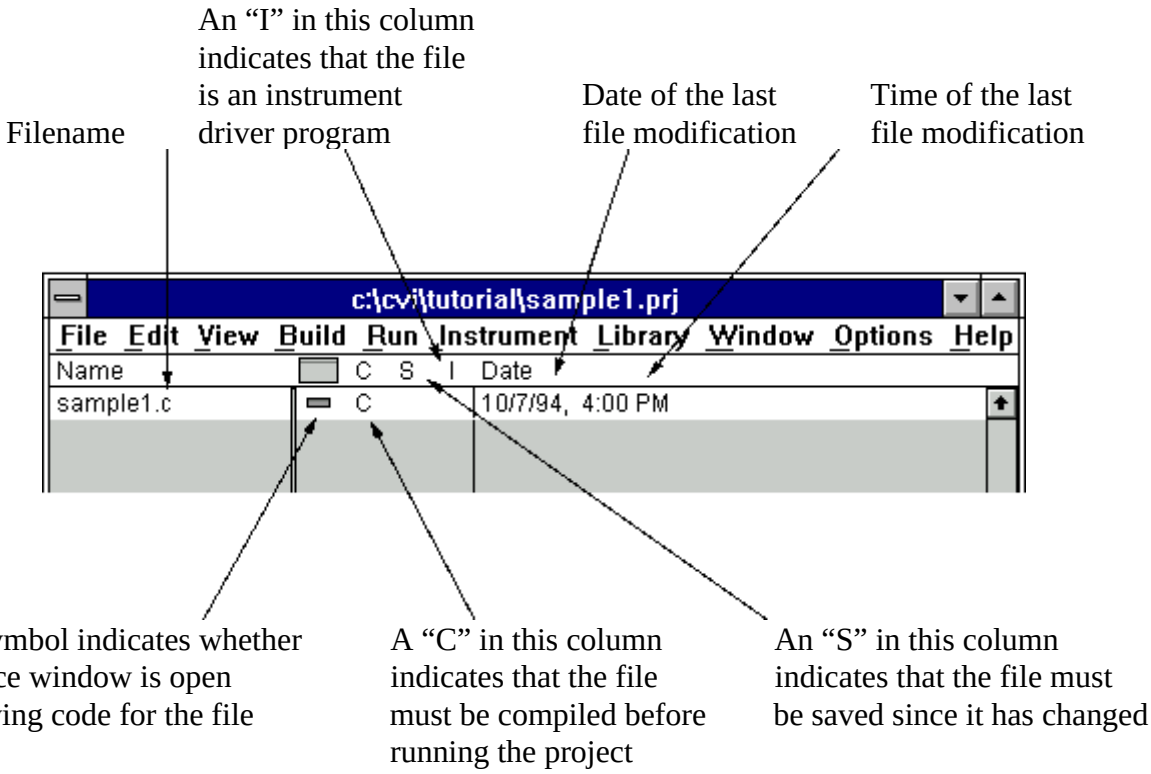
- The **File** menu contains selections for loading, saving, and creating new files in LabWindows/CVI. You can open any type of file (project, source, header, or user interface) from the **File** menu in the Project window.
- The **Edit** menu contains selections for adding or removing files from the 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, *The 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 the following illustration.



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. The following illustration shows how this information is displayed in the Project window.



Running the Project

To run the `sample1` project, select **Run Project** from the **Run** menu. LabWindows/CVI will automatically compile any source files in the project list (notice that the `C` indicator disappears

from the project list), link the project with the libraries used, and execute the compiled code. When the project begins running, the word **Running** will appear 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 is displayed. The error message window contains the number of errors detected in each source file and a description of the current error. For example, you can get an illegal character 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

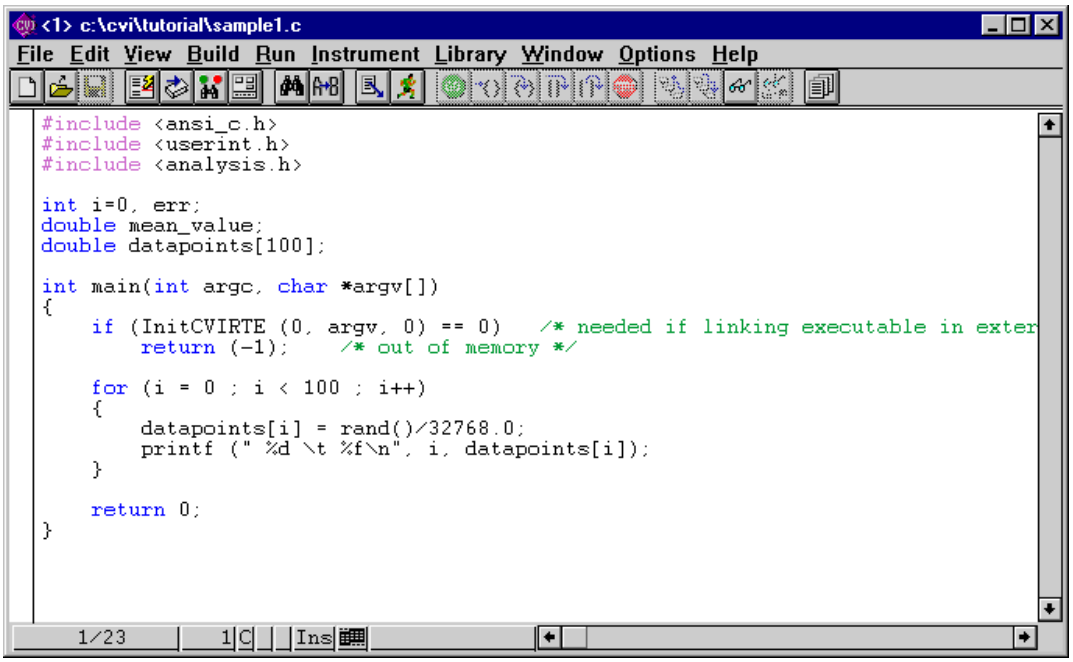
The Standard Input/Output window is where simple, text-based information is displayed to or received from the user during program execution. If you want to use the ANSI C `stdio` library when developing your C programs in LabWindows/CVI, the results of the `printf` and `scanf` functions appear in the 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 the following illustration. As you can see, the source code for the `sample1` file contains standard ANSI C compatible code.



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, *The 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 will be displayed 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 view 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.

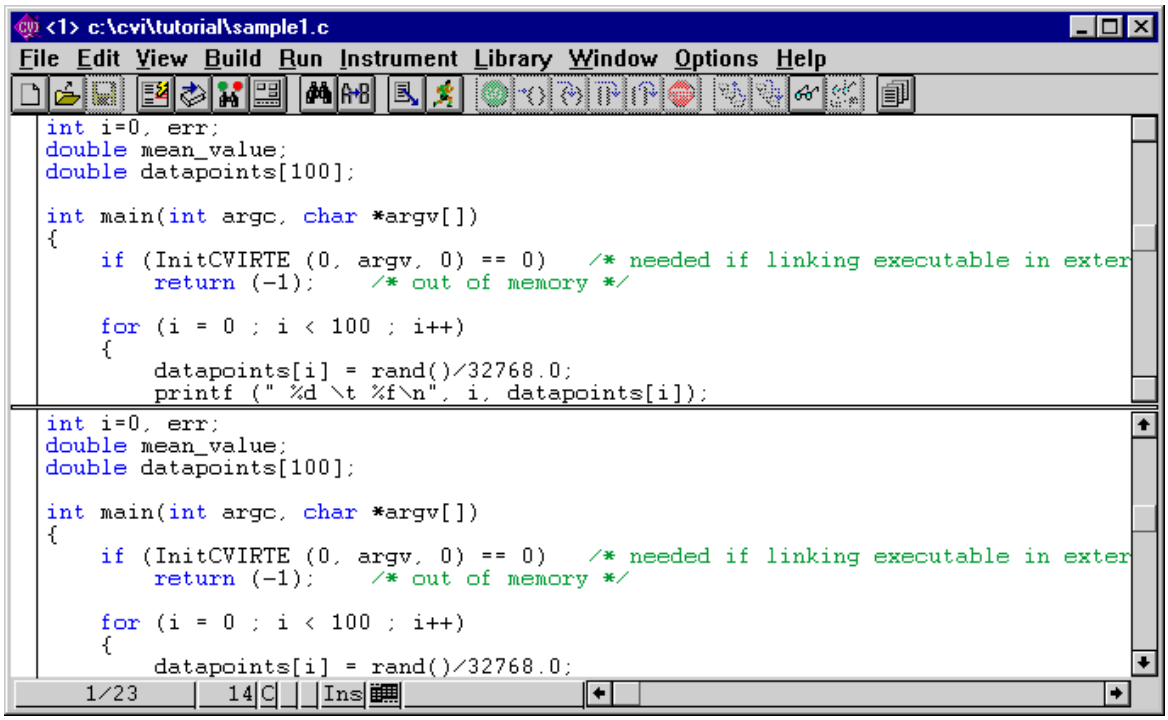
1. If you are viewing a large file, you may need to refer to particular line numbers. Select **Line Numbers** from the **View** menu. A new column will appear to the left of the window with line numbers displayed.

2. Many times, the programs you develop in LabWindows/CVI will 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 do this, grab the double-line at the top of the source window with the mouse and drag it to the middle of the screen. You should see a duplicate copy of the source code in each subwindow as shown in the following illustration.



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. There are two different methods 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 will appear in the left-hand column of the source window.

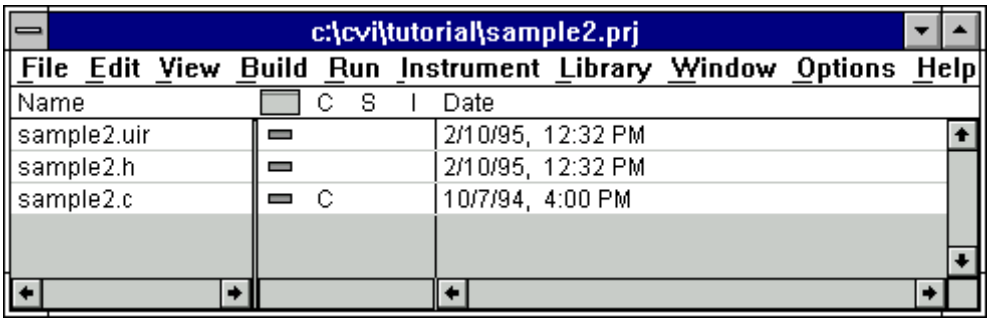
8. Move the cursor to line 12 of the Source window and enter another tag. By selecting **Next Tag** from the **View** menu, your cursor will automatically jump to the next tagged line in your source code. You can also jump between tags by pressing the <F2> key.

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

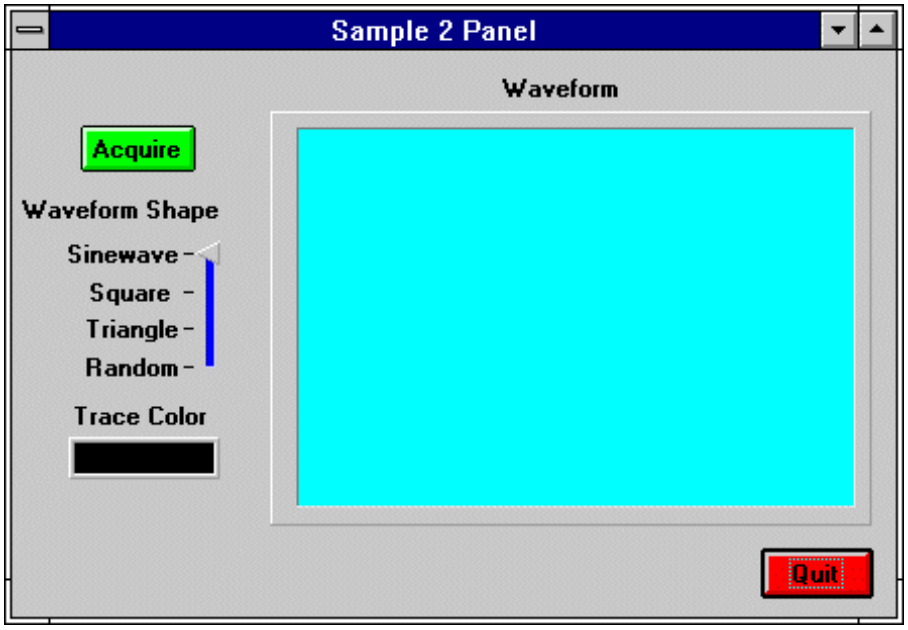
Operating Projects with a User Interface

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 may be prompted to save changes from `sample1.prj`. Select **Discard** to continue. The Project window should appear as shown in the following illustration.



2. Run the project by selecting **Run Project** from the **Run** menu, or by entering <Shift+F5>. The following GUI should appear once the program compiles and runs.



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. Now click on the color bar to

choose a new color for the waveform trace. Finally, click on the **Acquire** button once 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, you will learn how to use interactive code generation tools.

Chapter 3
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.

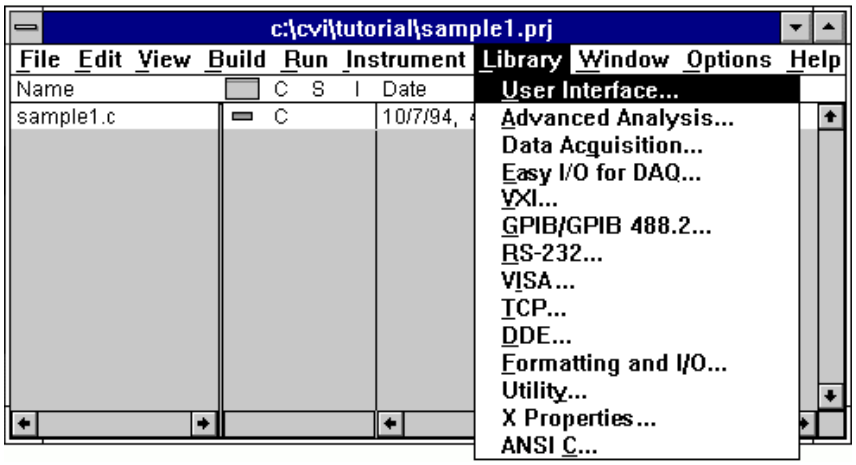
If you have not saved the previous contents of the windows, you will be prompted to do so. If you want to save the previous contents, click on the **Save** command button and enter a filename into the File Name input box. If you do not want to save the previous contents, click on the **Discard** command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

Now let's look at the **Library** menu.

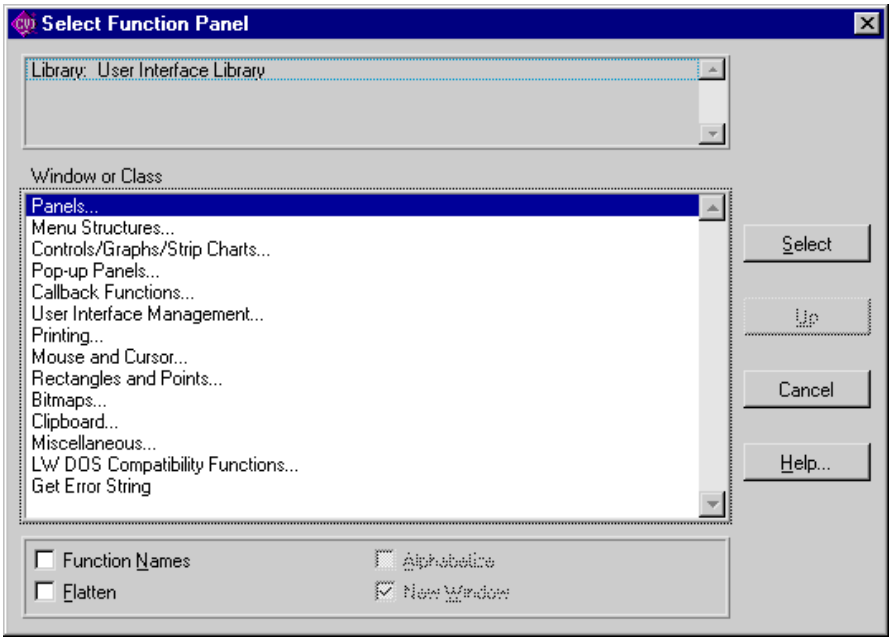
The Library Menu

All of the libraries in LabWindows/CVI are divided into a hierarchical structure under the **Library** menu as shown in the following illustration. 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. The VXI Library is accessible only by LabWindows/CVI VXI Development System users.***



When you select one of the libraries from the **Library** menu, you will bring up a function tree. 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 the following illustration.



Accessing the User Interface Library

You will now use the User Interface Library to display a graph of the random numbers 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.

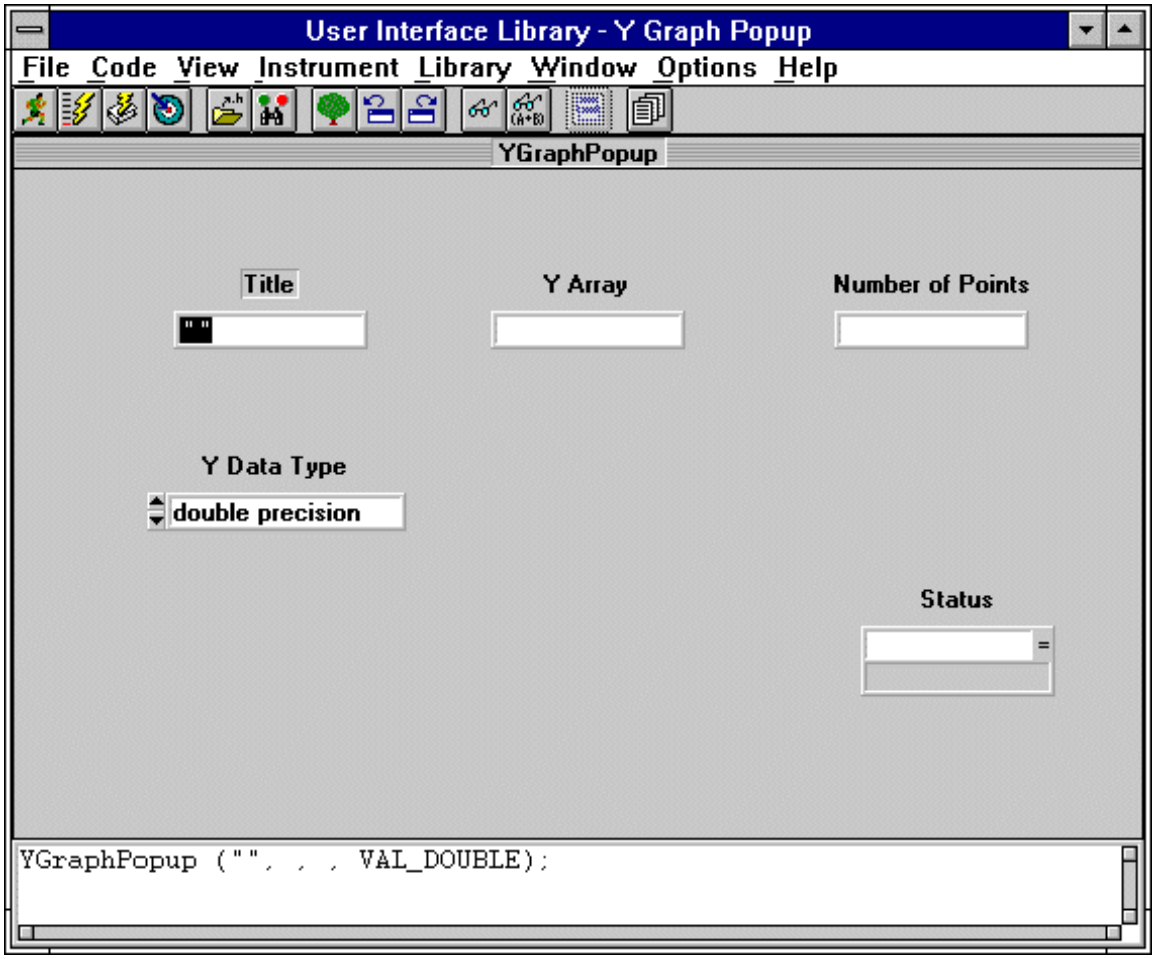
4. Position the input cursor before the line "`return 0;`" in the source file (line 19). This line is located before the closing right brace (`}`) for the `for` loop.

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) and 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 the following illustration.



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

There are two types of help information 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.

Panel Help Display Procedures

To Display This Type of Help …	**Perform This Action …**
Function Help	Select **Function** from the **Help** menu or Right click anywhere on the Function panel.
Control Help	Highlight the control, then select **Control** from the **Help** menu or 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.)

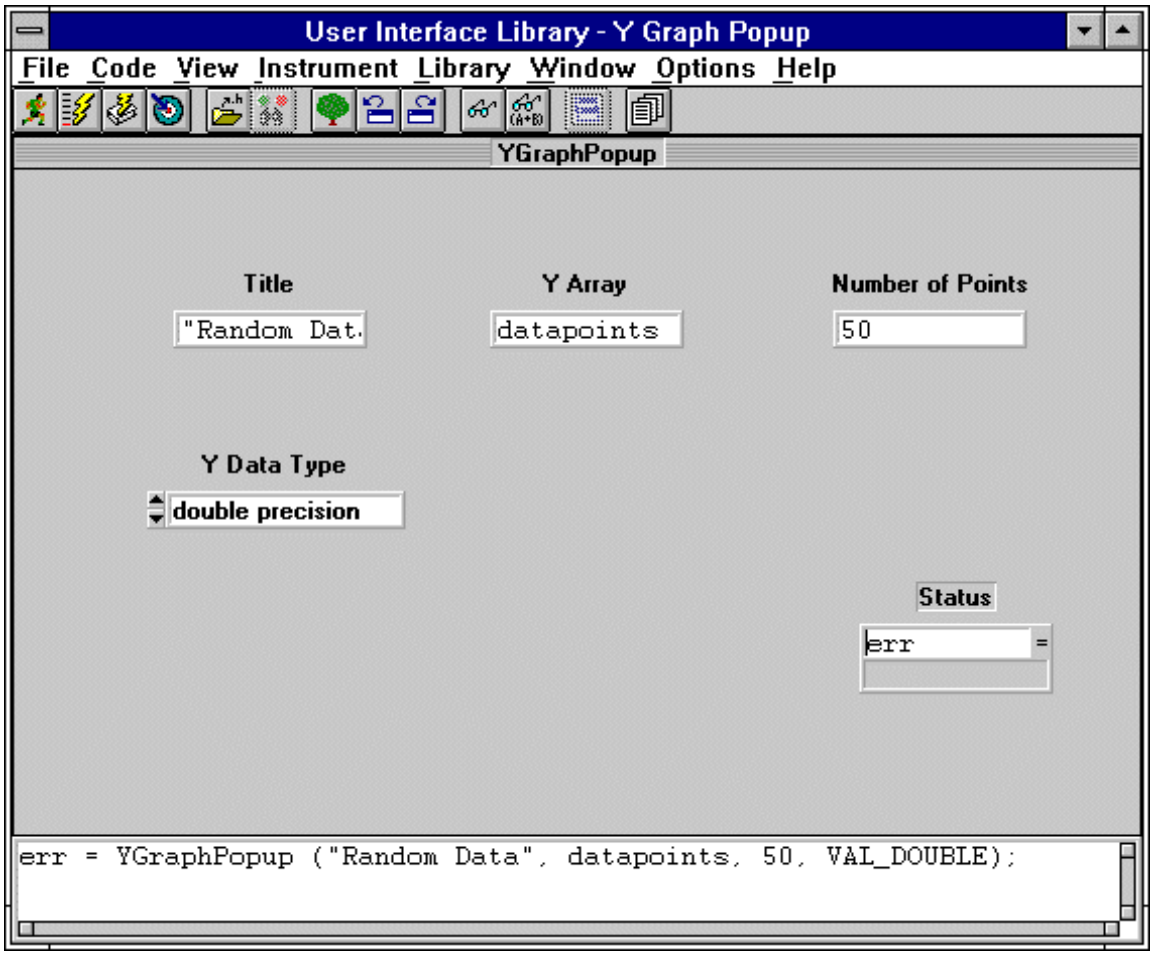
1. Enter "`Random Data`" in the Title control. (Be sure to include the quotes.)
2. Press <Tab> to move the highlight control to the Y Array control.
3. Enter the array name `datapoints` in the Y Array control.
4. Press <Tab> to move the highlight to the Number of Points control.
5. Enter the number `50` 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`. With a 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. (With a mouse, click on the Status control.)

9. Enter the variable name `err` in the control.

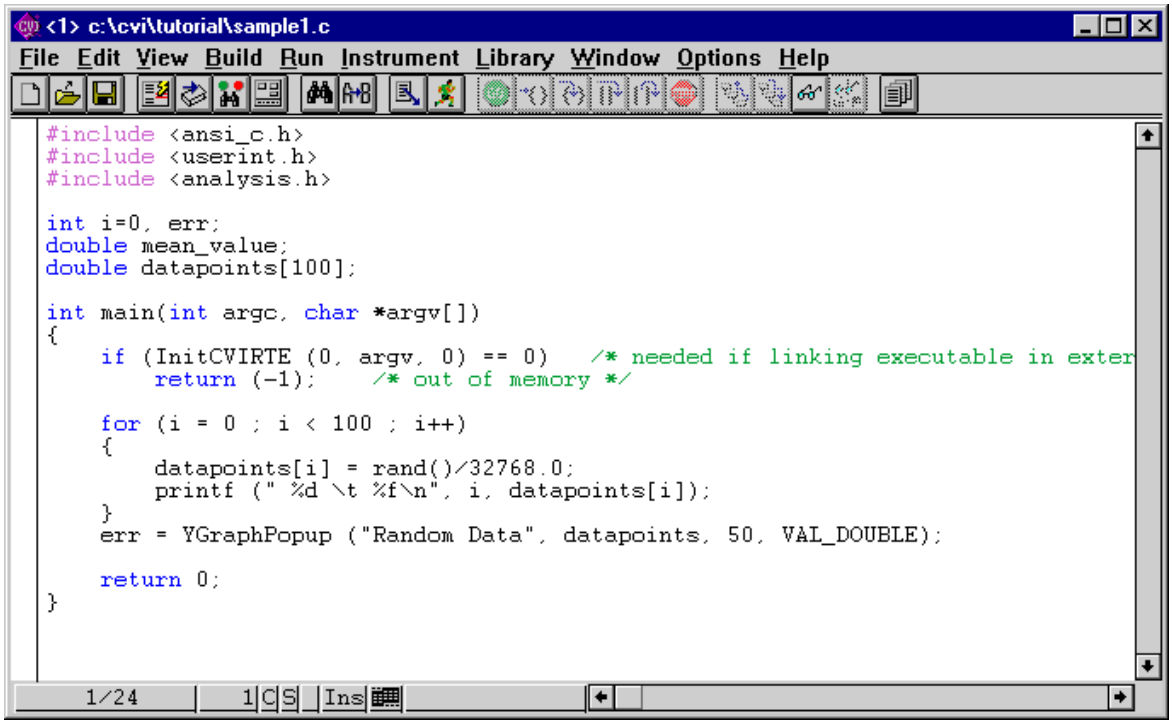
The function panel appears as shown in the following illustration.



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 going through 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.
4. The `sample1.c` source code should have the `YGraphPopup` function added at the bottom of the program, as shown in the following illustration. Notice the line of code inserted from the function panel.



5. Execute the program by selecting **Run Project** from **Run** menu. Click on **Yes** if you get 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.

6. Close the Standard Input/Output window.

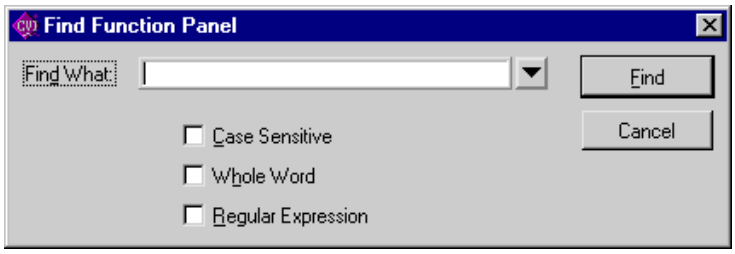
Analyzing the Data

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.

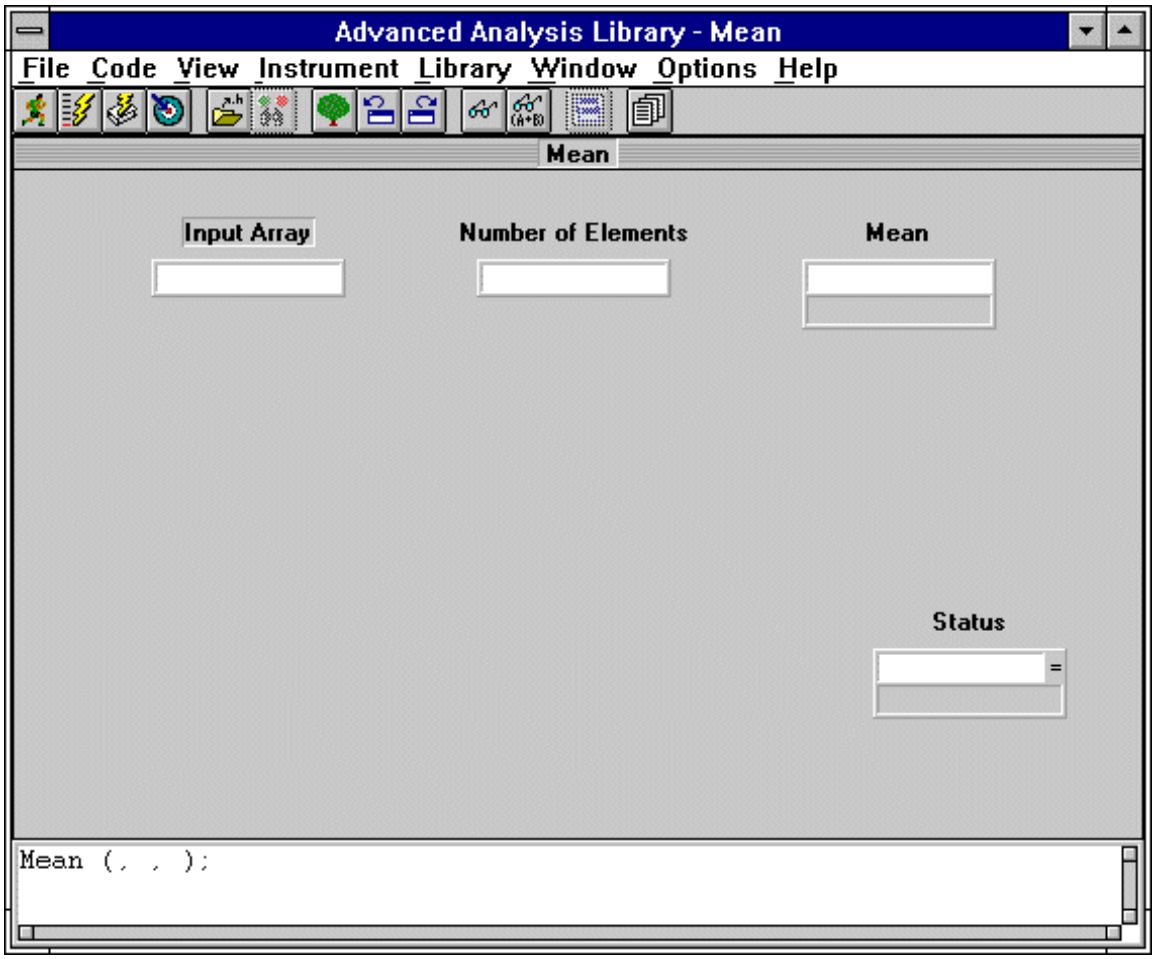
1. Select **Find Function Panel** from the **View** menu. A dialog box appears as shown in the following illustration.



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 the Mean function is found, its function panel appears, as shown in the following illustration.



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.

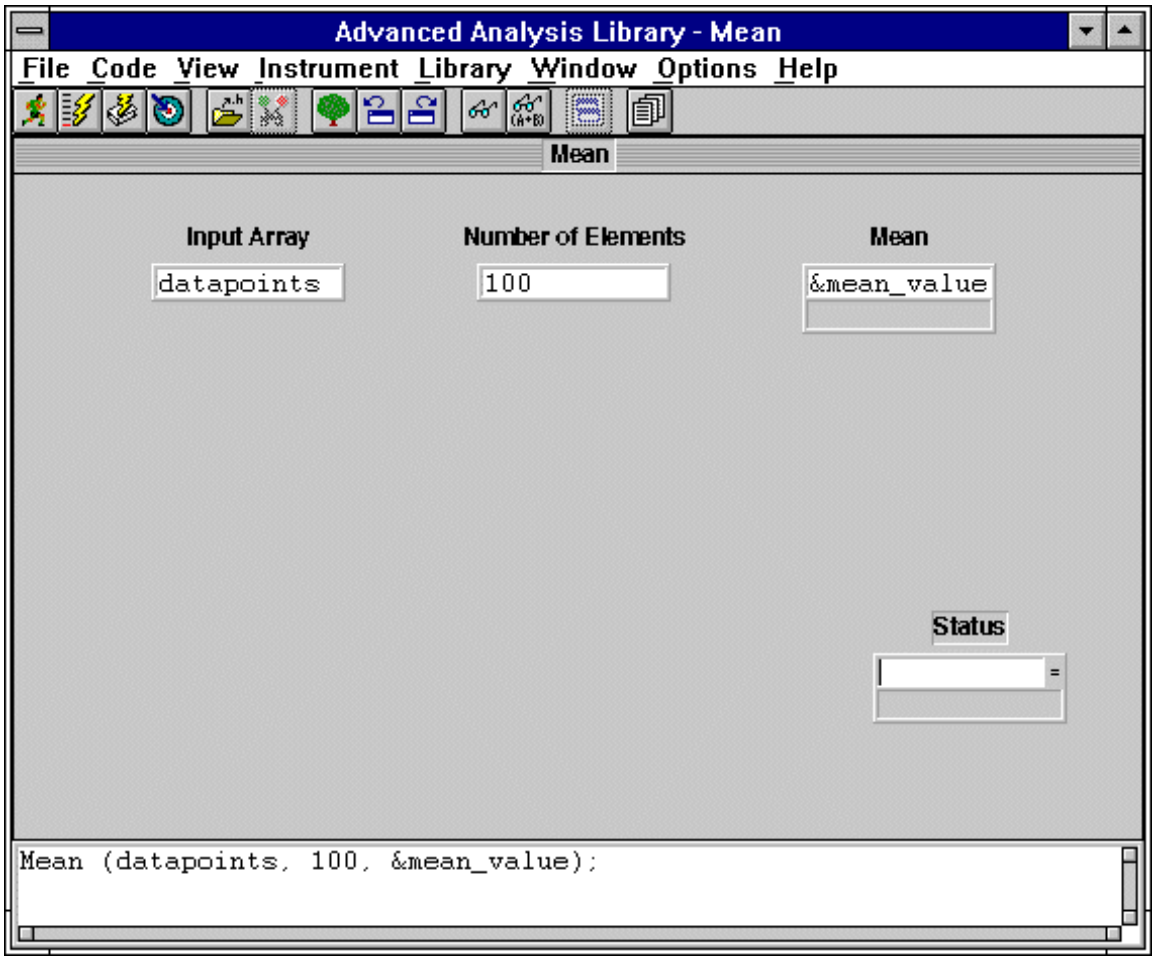
Output Values on a Function Panel

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. You will want to 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 the following illustration.



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.

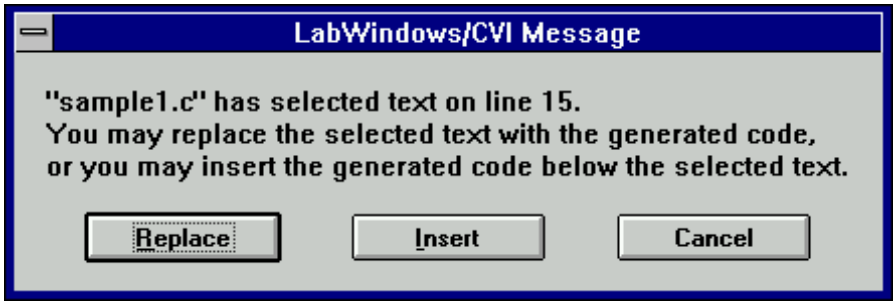
```
err = YGraphPopup ("random data", datapoints, 50, VAL_DOUBLE);
```

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 the following illustration.



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)   /* needed if linking executable in
external compiler; harmless otherwise */
        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 Chapters 6 and 7 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.

Chapter 4
Executing and Debugging Tools

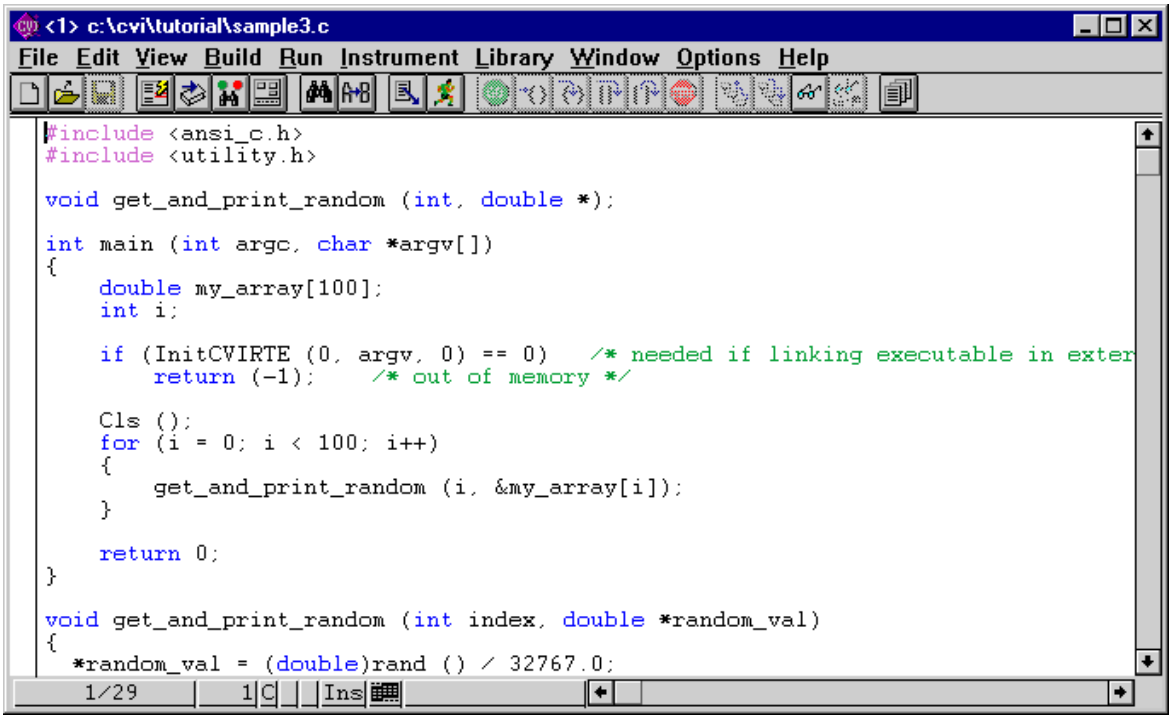
In this session, you will get acquainted with some of the tools available for executing and debugging in the LabWindows/CVI interactive program. This session describes the step modes of execution, breakpoints, the Variable Display, the Array Display, the String Display, and the Watch window.

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 the following illustration.



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 (Chapter 2).

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** will execute 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, and clicking 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 will be 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. There are four ways 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 this section. For additional information on Conditional Breakpoints and Breakpoint on Error, see the *Run Menu* section of Chapter 4, *The 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.

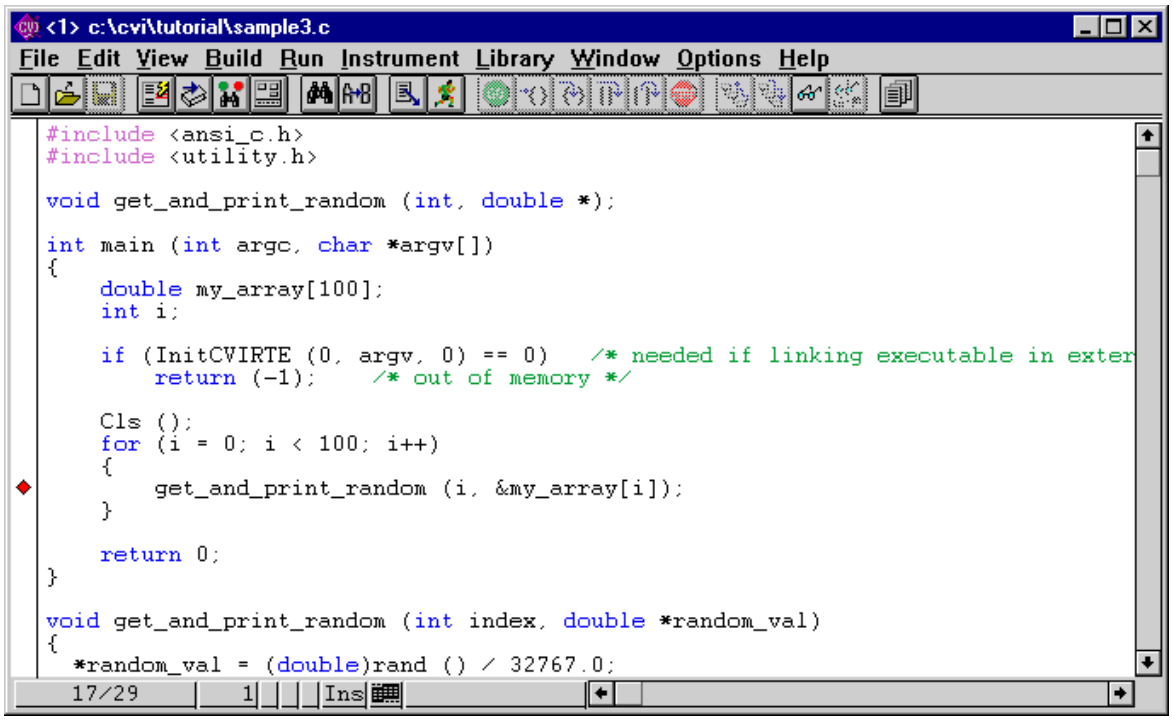
1. Halt program execution, if you have not already done so, by selecting **Terminate Execution** from the **Run** menu or clicking on the **Terminate Execution** toolbar icon.

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 the following illustration.



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 following events occur.

- Program execution is suspended.
- 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> (or <Ctrl-F12> under Windows 95/NT and UNIX), or select **Terminate Execution** from the **Program** menu to halt the program.

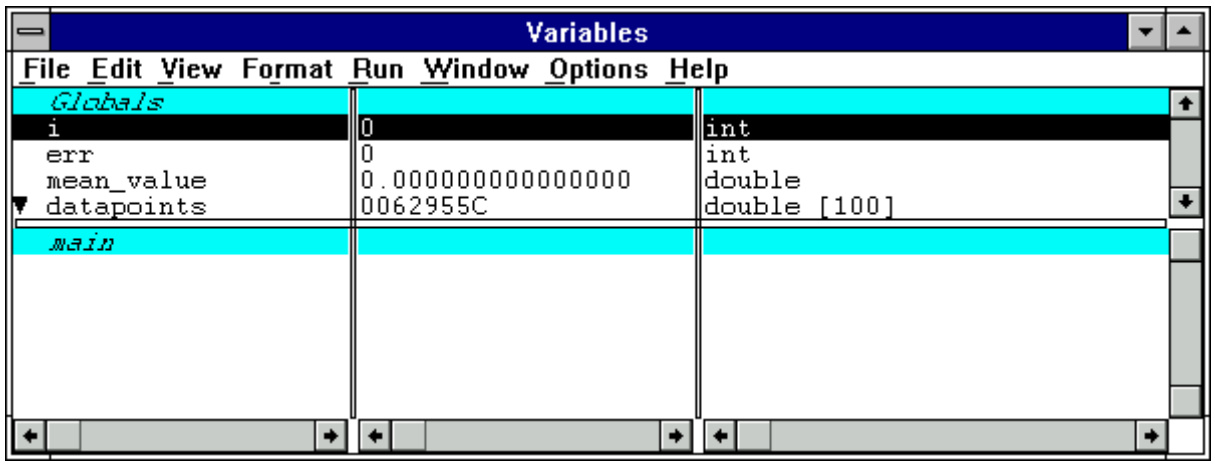
Displaying and Editing Data

Step mode execution and breakpointing are useful tools for high-level testing. There are many cases, however, when 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 Variable Display, 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. Each display is described in the following sections.

The Variable Display

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

When you select Variables, the display appears as shown in the following illustration.



The Variable Display 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,

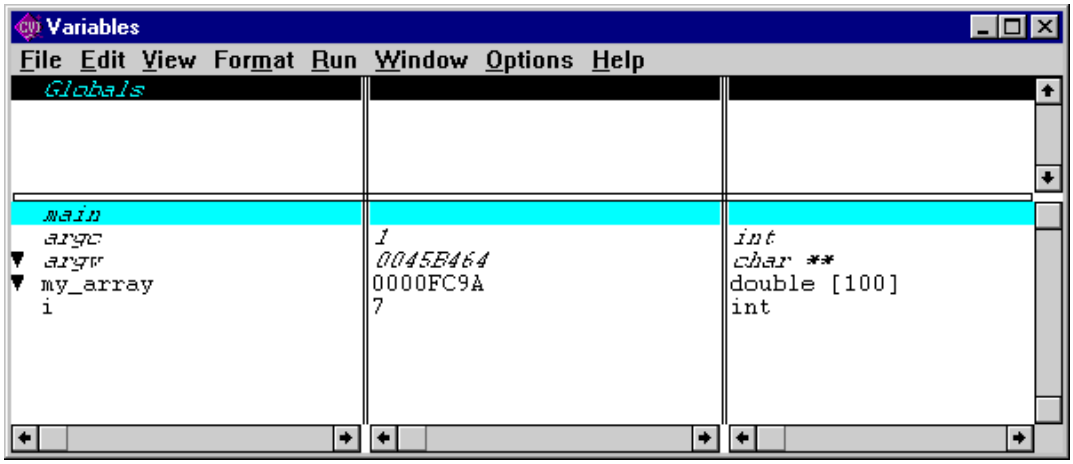
and in which file they are defined. The display in the preceding illustration shows that there are a number of global or local variables currently declared in two different source files.

You can view the Variable Display 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 Variable Display 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 Variable Display appears as shown in the following illustration.



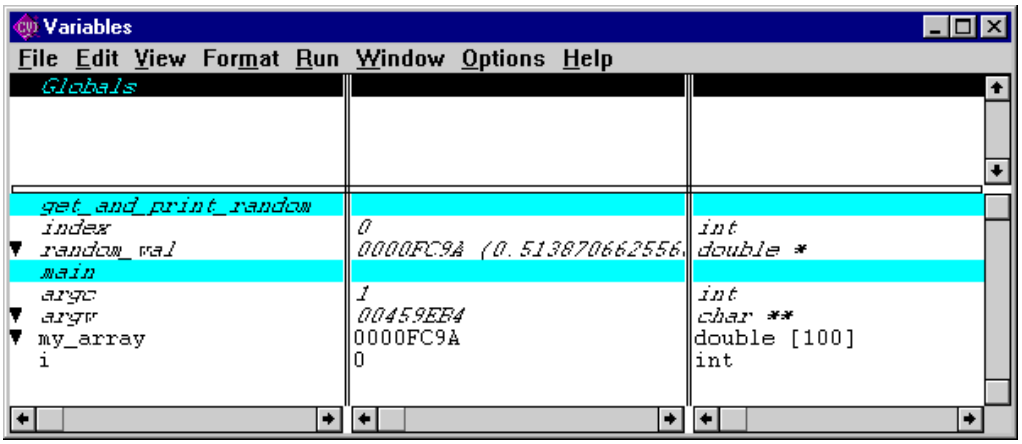
   Notice that there are now two variables—a double-precision array (`my_array`)—and an integer (`i`) under a heading called *main*. The section called *main* displays all variables that are declared locally to the main program in the **Program** window.

Note: ***The values may differ from the value 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> continuously 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.`

6. Select **Variables** from the **Window** menu. The Variable Display now appears as shown in the following figure.



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

7. Click 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 Variable Display. The Variable Display groups variables according to their scope (global or local) and the program module in which they are declared (main program, or subroutine).

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

Editing Variables

In addition to displaying variables, you can use the Variable Display to edit variable contents. The following steps illustrate the use of the Variable Display 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`.

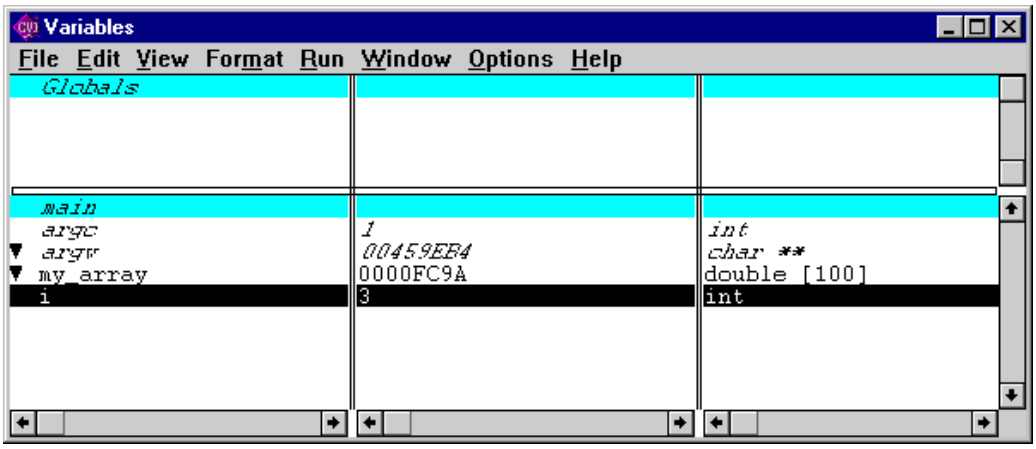
```
*random_val = (double)rand () / 32767.0;
```

Perform the following steps:

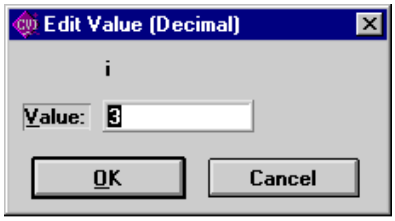
1. Select **Step Into** from the **Run** menu or press <F8> continuously until the for loop executes a few times and the highlight appears on the following statement.

   ```
   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.



3. Press <Enter> to display the 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> continuously.

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 Variable Display 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 Variable Display.

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

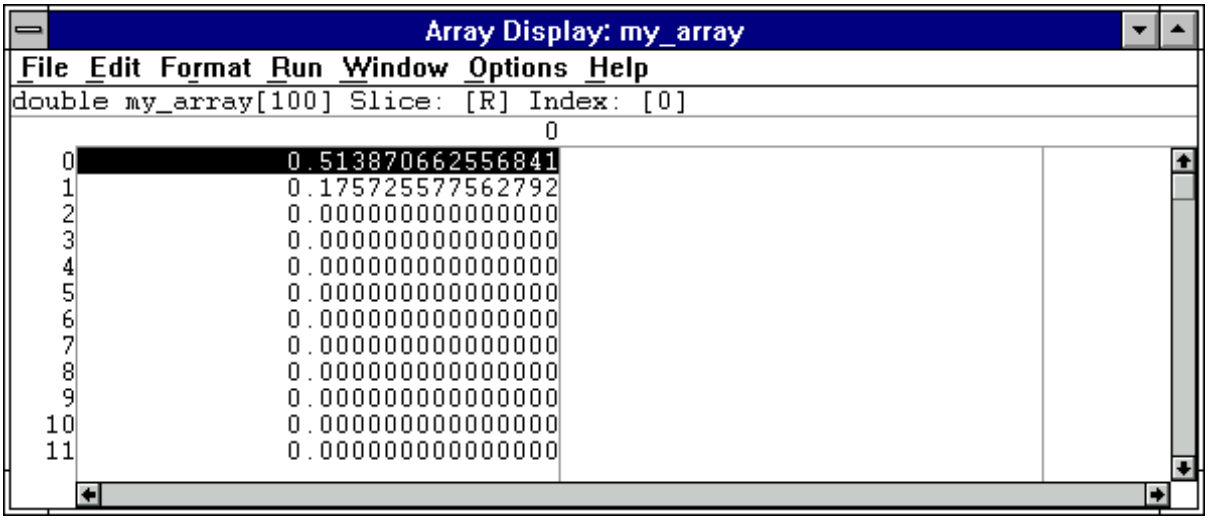
1. Click on the Source window to make it active.

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 Variable Display 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>. (With a mouse, double-click on the variable name `my_array`.) The display shown in the following illustration appears on your screen.



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.***

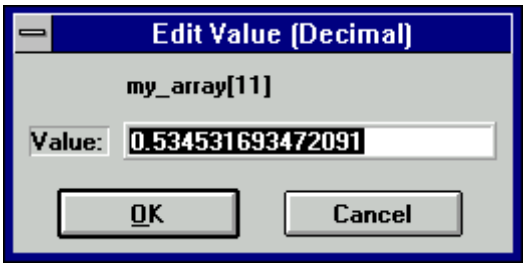
The display shown in the preceding illustration is the Array Display. The Array Display shows the values of array elements in tabular format. In this example, the array `my_array` is a one-dimensional array, so the display consists of one column of numbers. The numbers in the column on the left side 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 Variable Display. 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 the following illustration appears.



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 Variable Display.

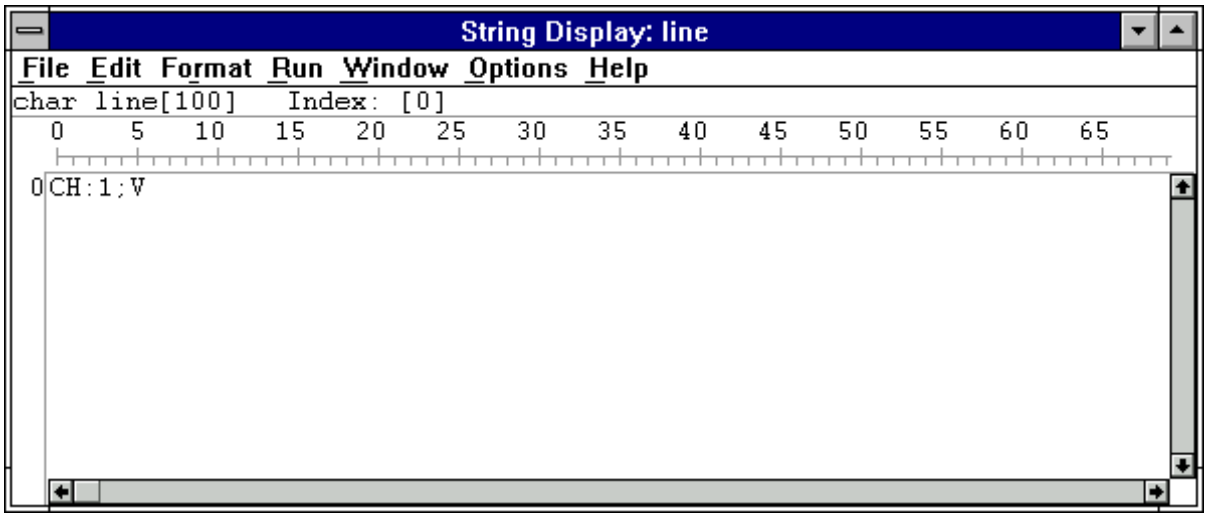
4. Click on the Source window in the background to make it the active window.

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 edited value, 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 Variable Display. 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, *The Array and String Display Windows*, in the *LabWindows/CVI User Manual*. An example String Display is shown in the following illustration.



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.

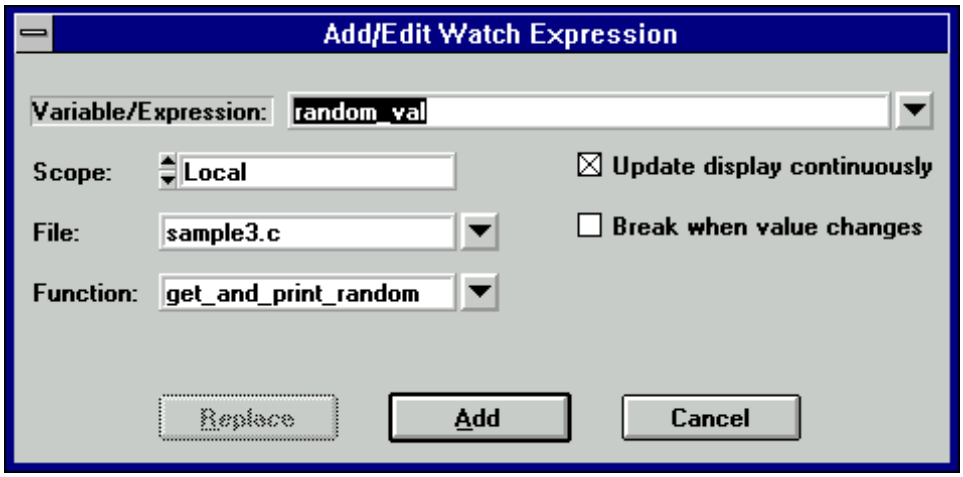
3. Press <F8> continuously to place the highlight on the following line.

   ```
   void get_and_print_random (index, random_val);
   ```

4. Select **Variables** from the **Window** menu to open the Variable Display, or click on the Variable display if it is already active in the background.

5. Click on the r`andom_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 shown in the following illustration will appear.



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.

Chapter 5
Building a Graphical User Interface

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 be 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.

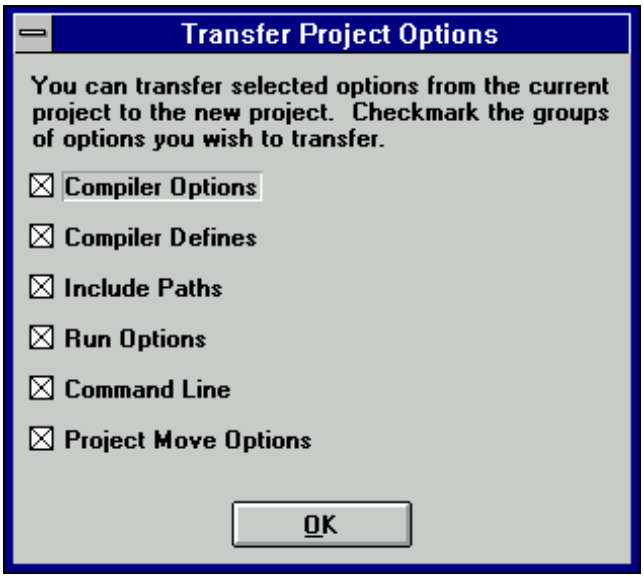
You can use the User Interface Editor to create a GUI for an application program. A user interface is comprised of a collection of objects such as menu bars, panels, controls, and pop-up menus. In the Chapter 6 tutorial session, you will be introduced to the User Interface Library, which includes a set 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 the following illustration should appear. Click on **OK** to maintain the current configuration of the LabWindows/CVI environment for the new project.



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.

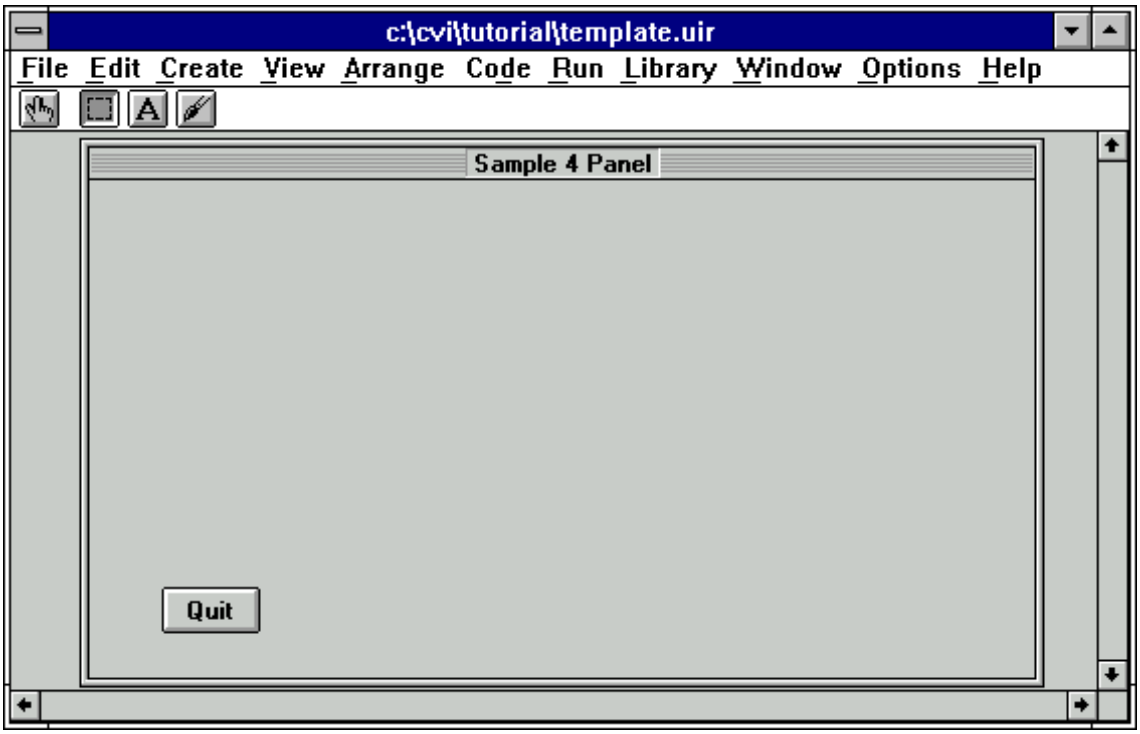
1. Create a User Interface in the User Interface Editor (Chapter 5).
2. Generate a shell program source file using CodeBuilder (Chapter 5).
3. Add to the C source code to generate and display the waveform (Chapter 6).
4. Develop your own callback function to compute the mean value of the waveform (Chapter 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

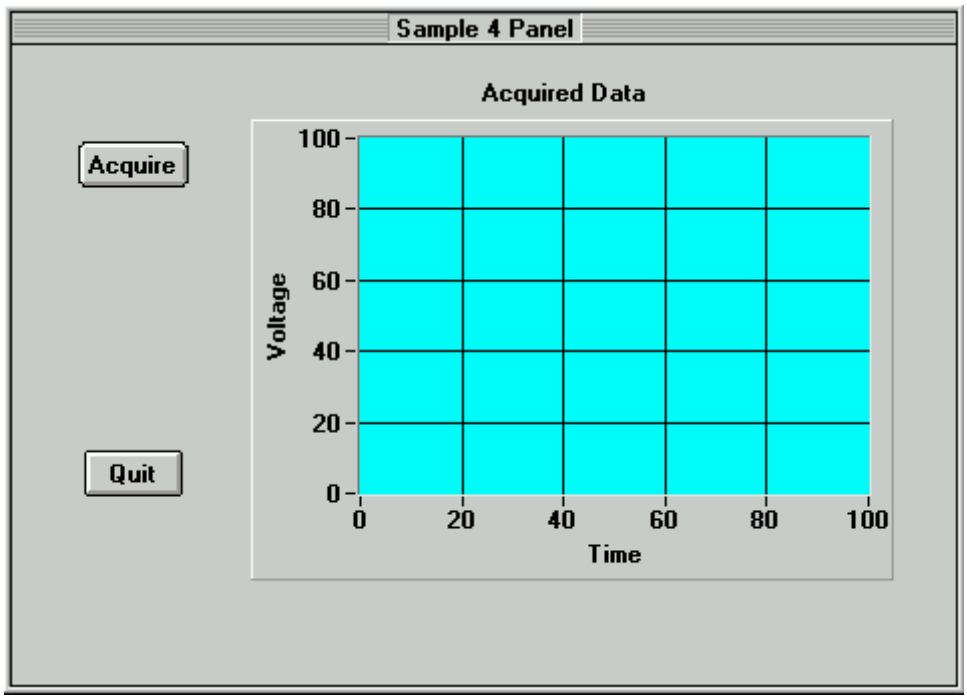
1. Select **Open User Interface** (`*.uir`) from the **File** menu. Select the file `template.uir` from the file list. The User Interface Editor will appear with the controls as shown in the following illustration.



The **Edit** menu contains selections to 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.

2. Notice that the user interface has been partially completed for you. 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 the following illustration. You will add the button and graph control in the steps that follow.

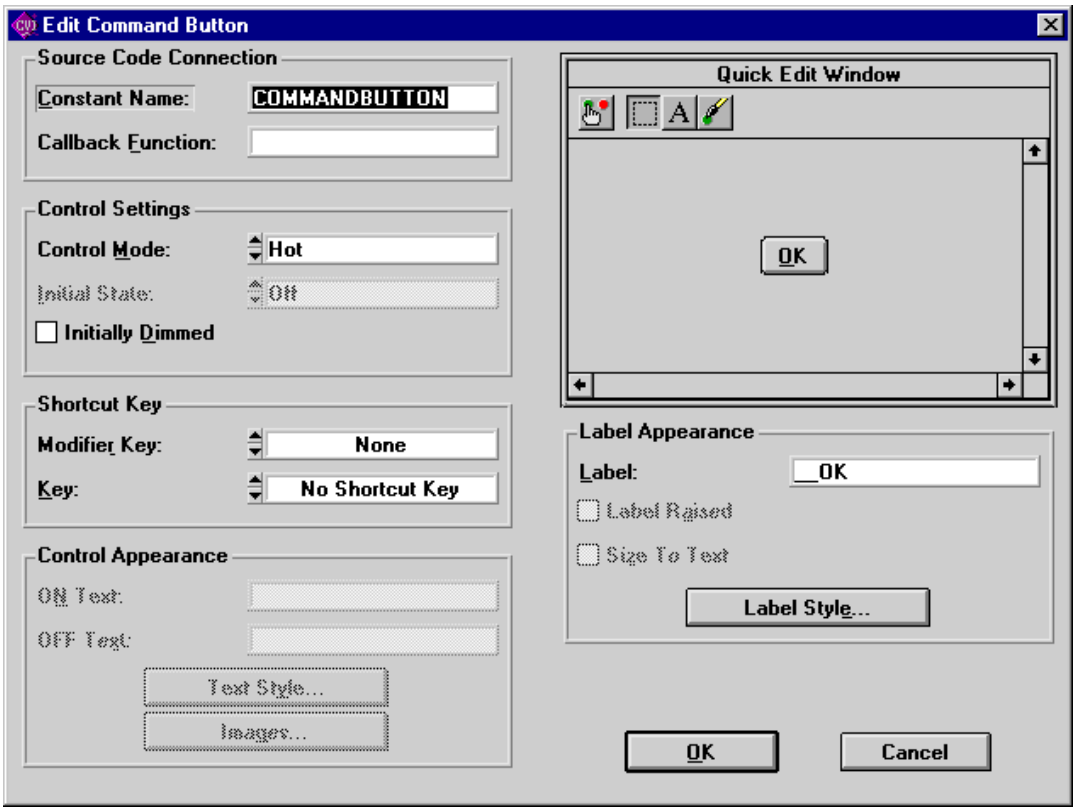


Step 2: Adding a Command Button

1. 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.

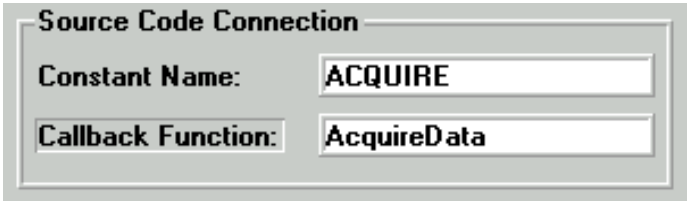
2. Use the mouse to position the button on the panel. If you do not have a mouse, press the <Tab> key until the **OK** button is highlighted, and then use the arrow keys to position the button.

3. 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 the following illustration.



4. 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 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.)

5. 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 the following illustration to ensure that your program can properly communicate with the button.

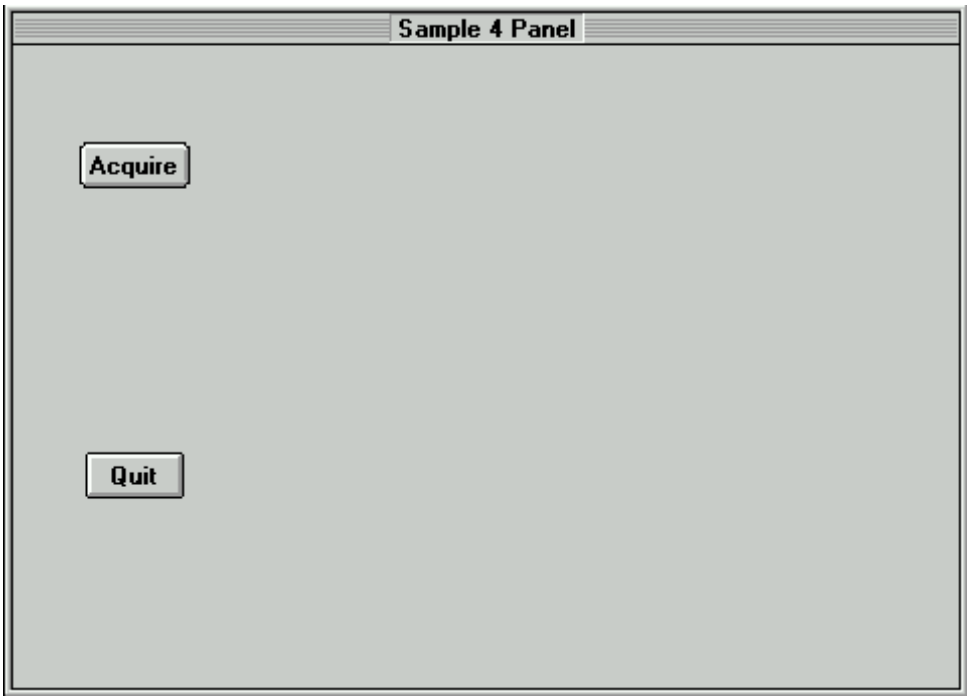


6. Press the <Tab> key five times, or click with the mouse, to highlight **OK** in the **Label** input within the **Label Appearance** section of the dialog box.

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 bring up 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 the following illustration.



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.

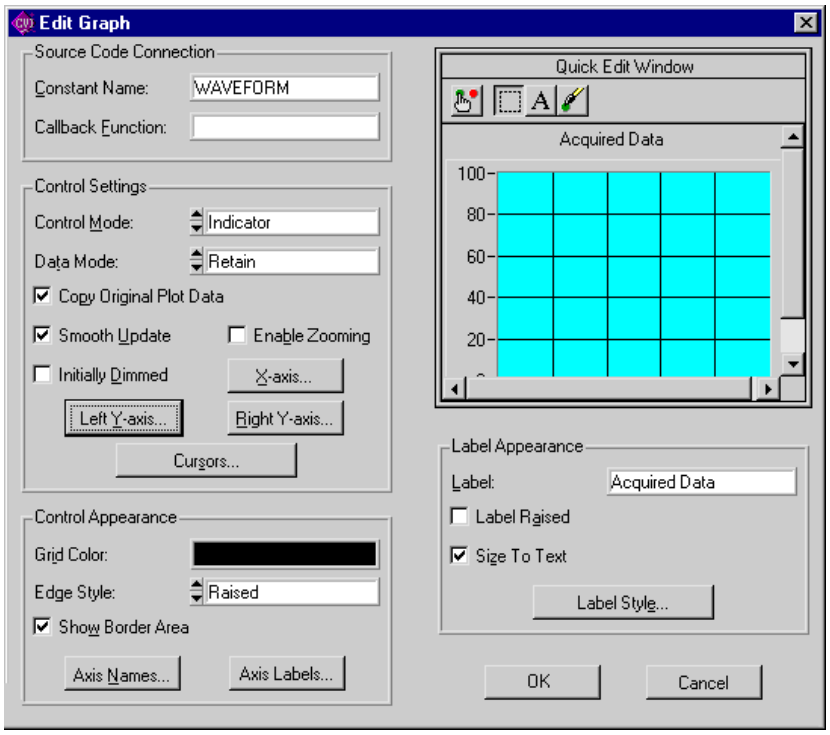
2. Position and size the graph (drag one of the corners) with the mouse.

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: ***Since 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 the following illustration.

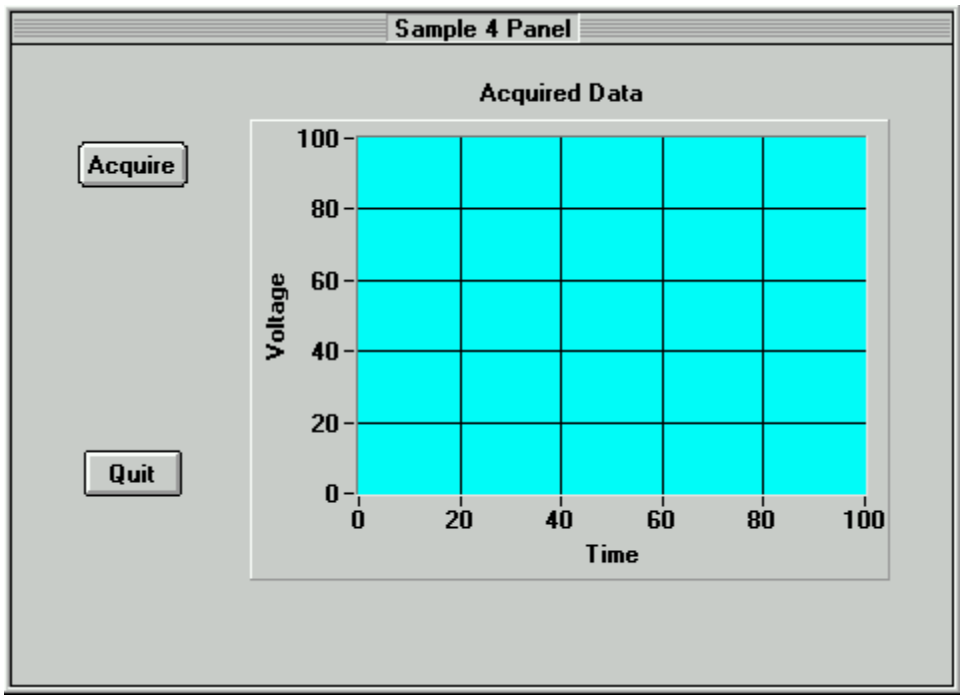


6 [Optional] Use the **X-axis** and **Left Y-axis** buttons to display the Edit Axis Settings dialog box. Assign `Time` and `Voltage` labels to the X and Y axes labels respectively.

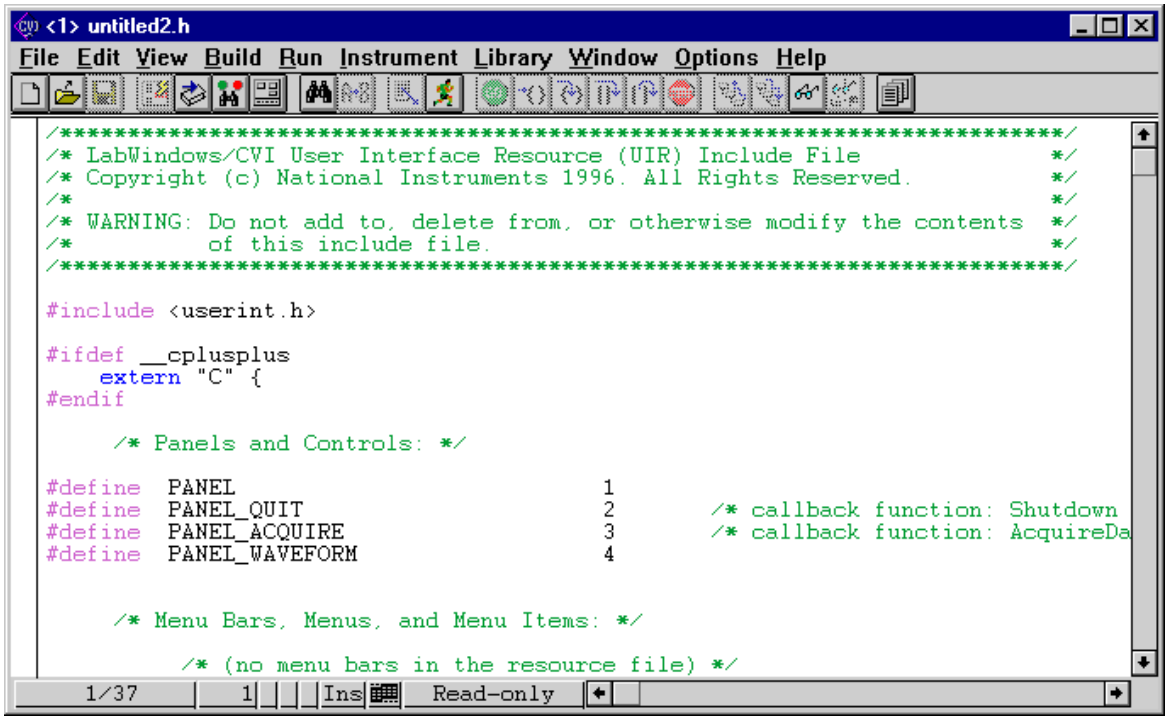
7. Once you have finished setting the graph attributes, click on the **OK** button at the bottom of the Edit Graph dialog box to close the dialog box.

Step 4: Saving the .uir File

1. Your completed user interface should look like the one shown in the following illustration.



2. Select **Save As** from the **File** menu to save the `.uir` file with the new controls added.
3. 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 follows.

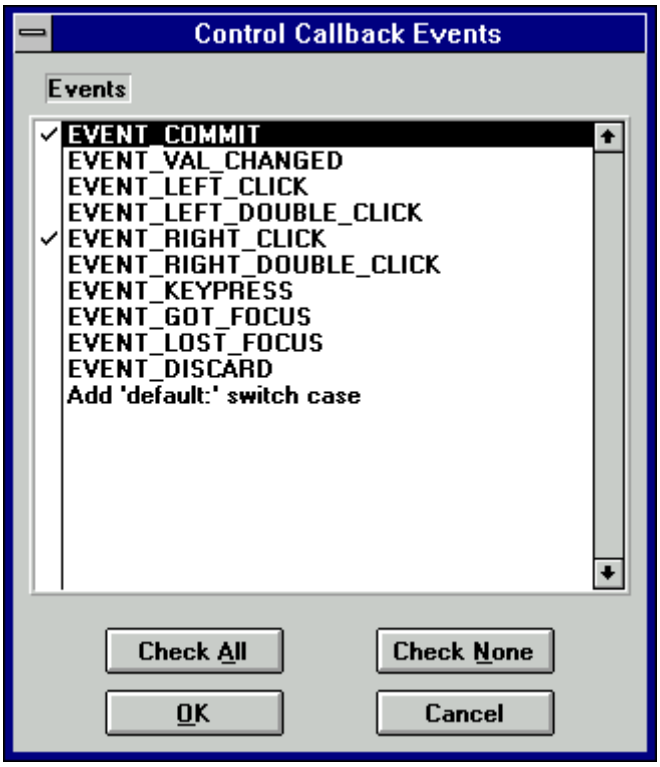


5. Close the header file before continuing by selecting **Close** from the **File** menu. You do not need to save it.

Step 5: Generating the Program Shell with CodeBuilder

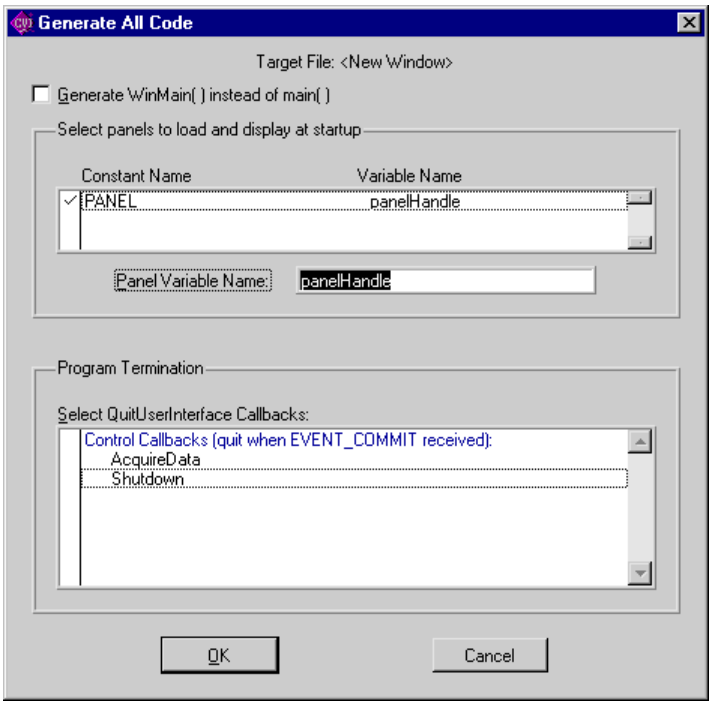
Now that you have created a GUI, CodeBuilder can help you get a jump start on development by automatically generating the start of your source code.

1. First, you must specify which events for which your program will respond. Select **Default Control Events** from the **Preferences** selection of the **Code** menu.



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 event list 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 following 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*: Make sure that the Panel Variable Name is `panelHandle` for this exercise.*

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>  /* needed if linking executable in external compiler;
                        harmless otherwise */
#include <userint.h>
#include "sample4.h"

static int panelHandle;

int main (int argc, char *argv[])
{
   if (InitCVIRTE (0, argv, 0) == 0)      /* needed if linking executable in
                        external compiler; harmless otherwise */
      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.

Chapter 6
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 click on it with the mouse 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 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)   /* Needed if linking in external
                                           compiler; harmless otherwise */
        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

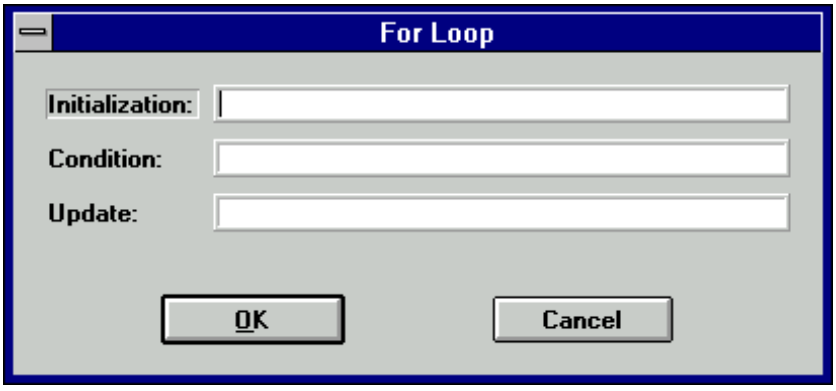
Your task is to complete the source code for `sample4.c` so that the program generates an array of random numbers and plots the array on the graph control. Most of the action will take place in the `AcquireData` function. When a user clicks on the **Acquire** button, the program will generate a random 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 following dialog.



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.

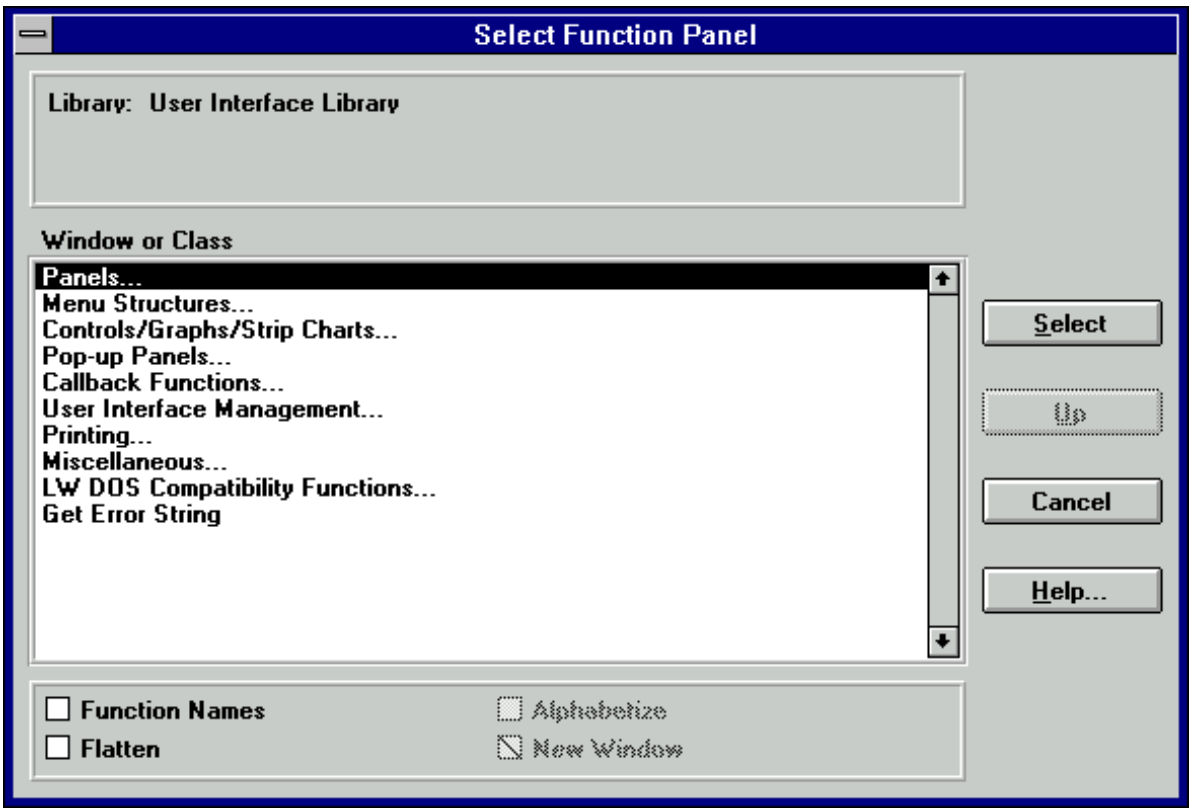
```
datapoints[i] = rand()/32767.0;
```

Finding the PlotY Function

Follow these steps to generate a line of code that will plot 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.

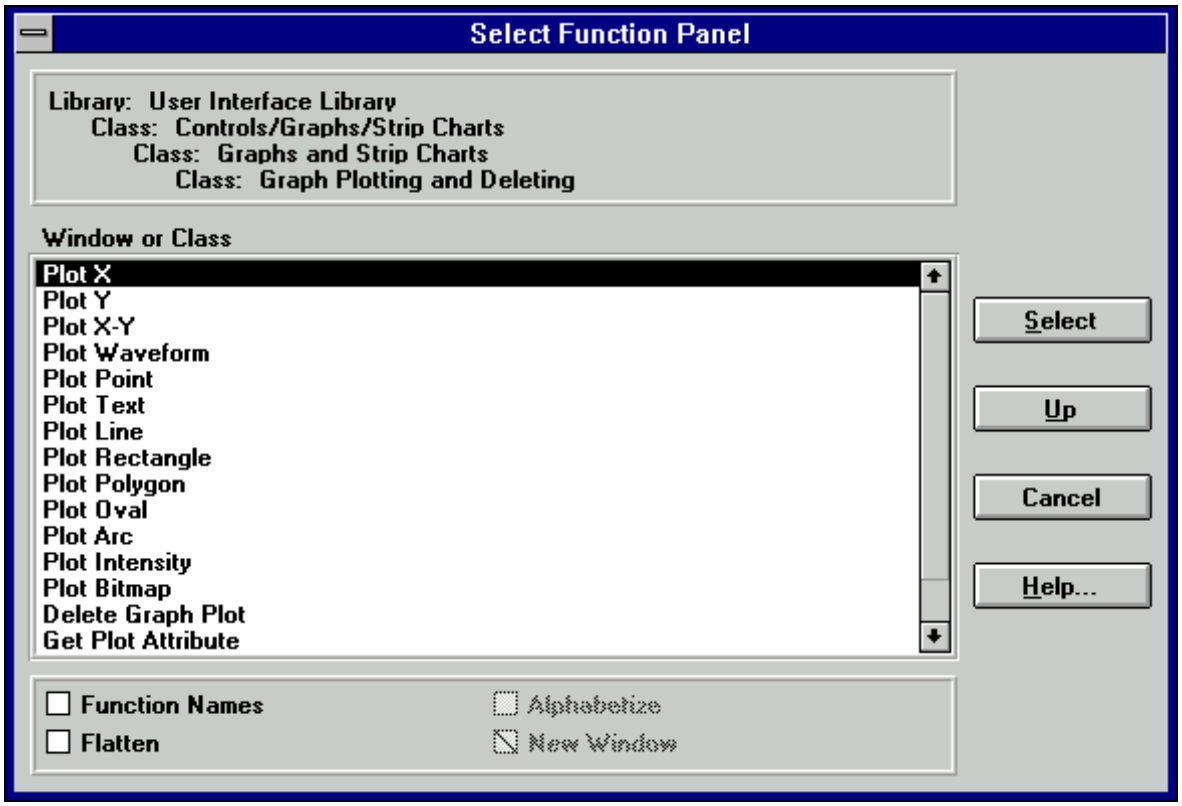
2. Pull down the **Library** menu and select **User Interface...** to display the dialog box shown in the following illustration.



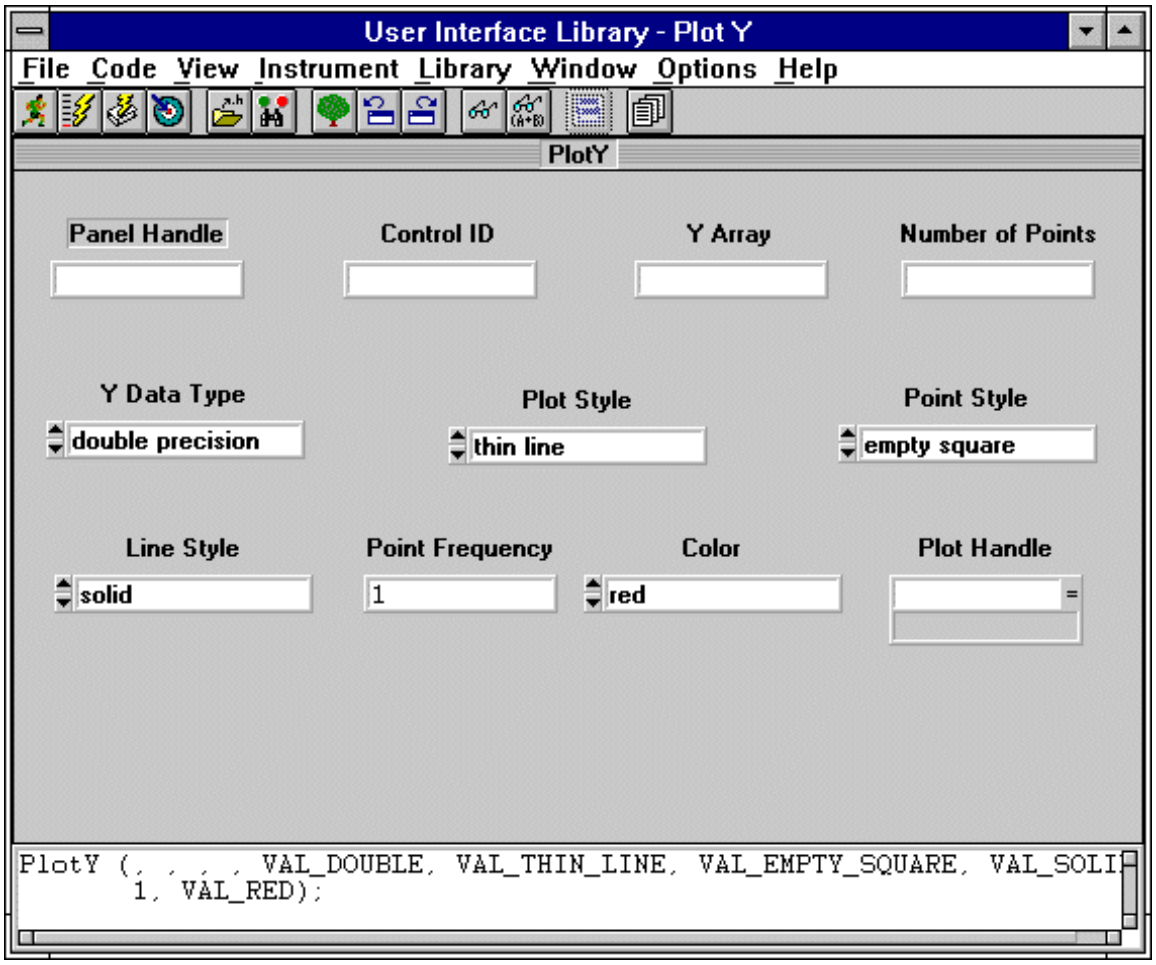
3. Use the <Down> arrow key to highlight the **Controls/Graphs/Strip Charts** selection and press <Enter>.

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 the following illustration.



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 the following illustration.



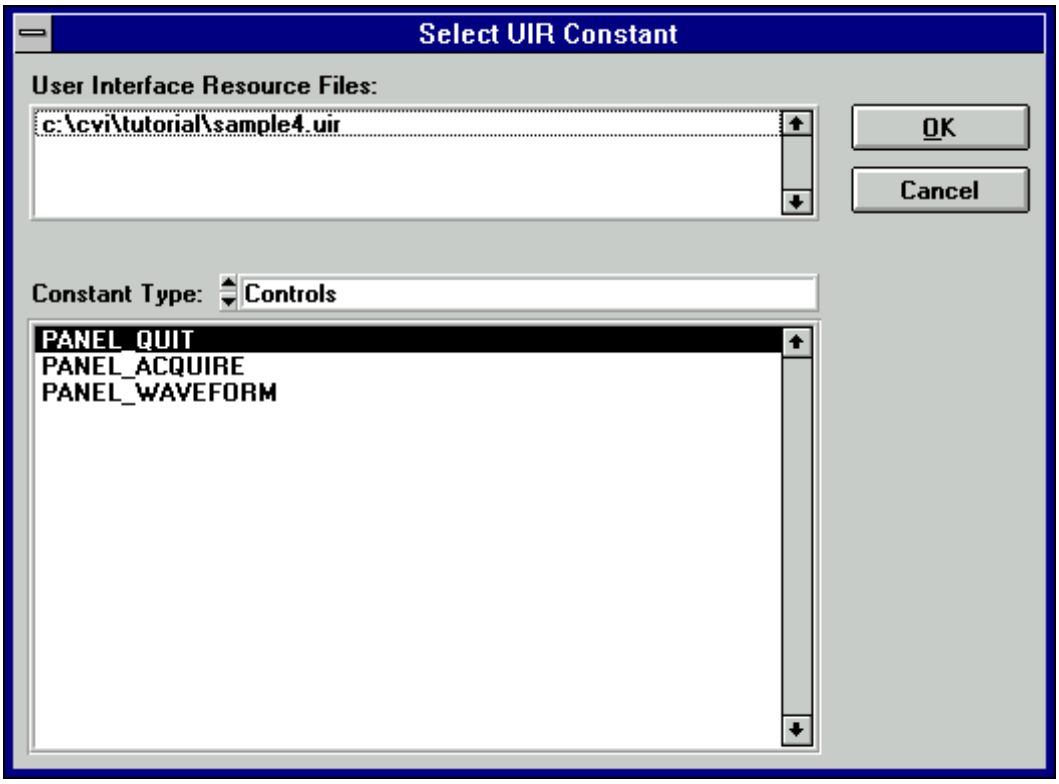
Building the PlotY Function Call Syntax

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 follows.

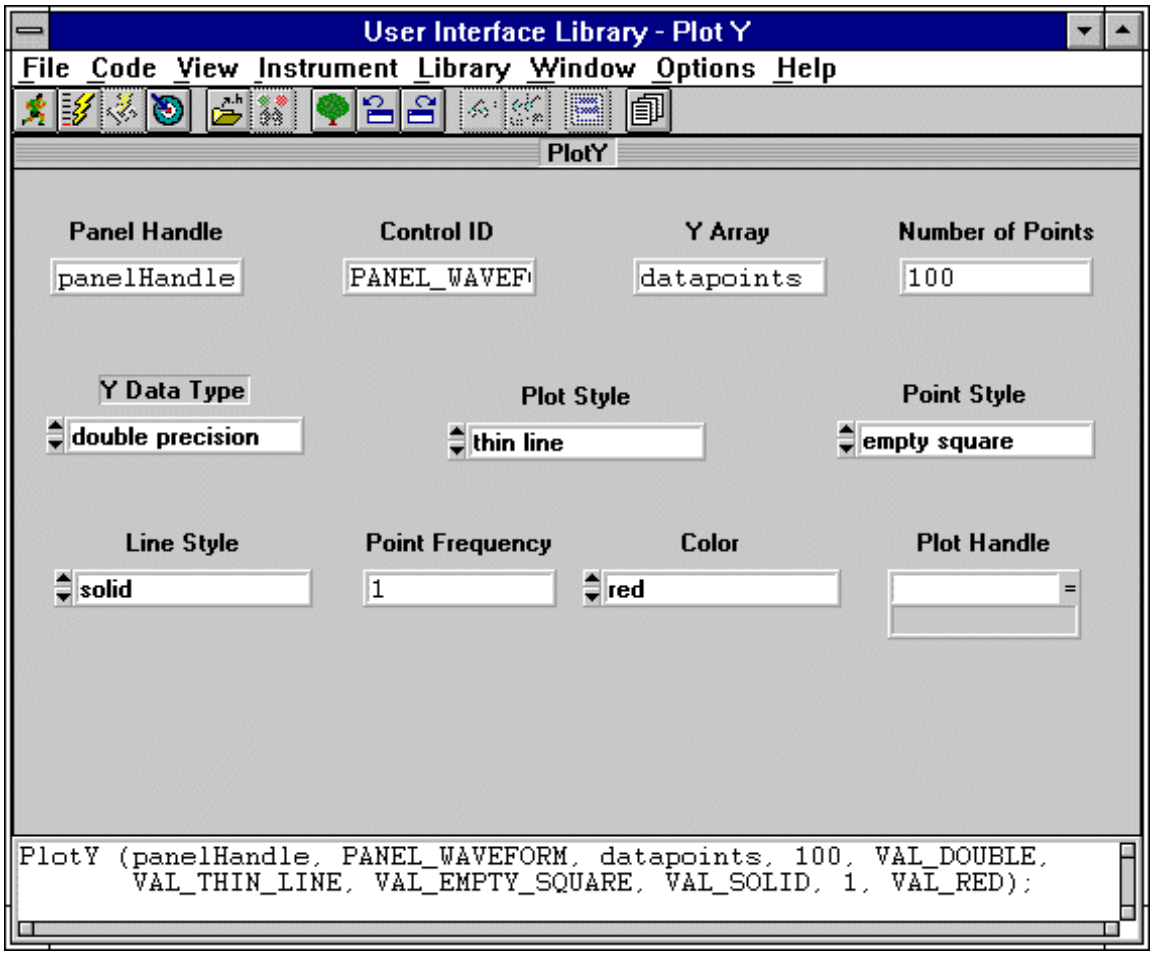


Select `PANEL_WAVEFORM`. Press the **OK** button to continue.

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.

5. You have completed generating the source code using the `PlotY` function panel, which should look like the following illustration.

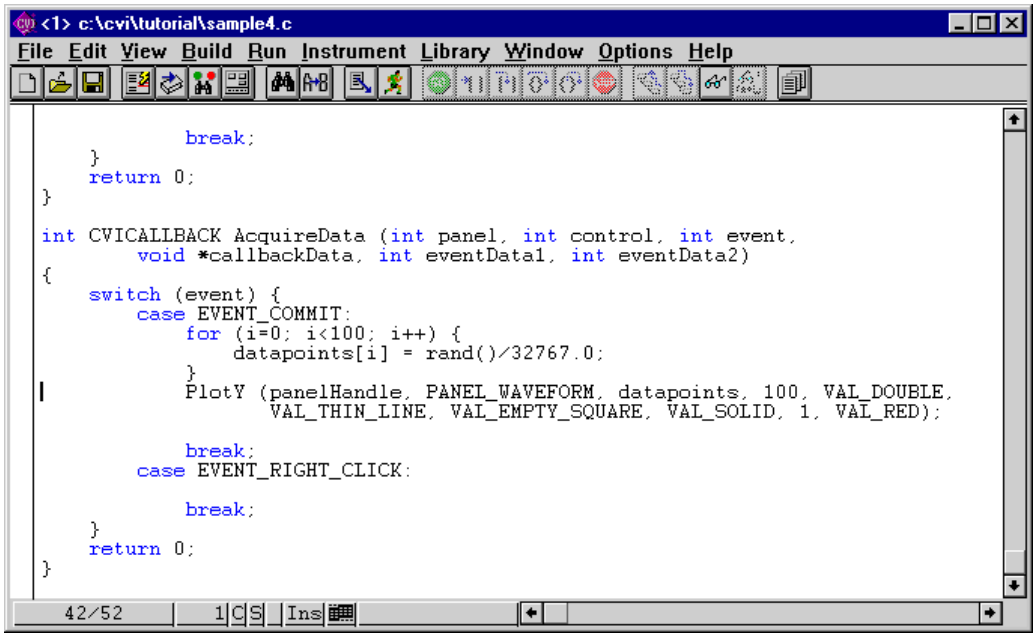


6. Select **Set Target File** from the **Code** menu to indicate the window in which the function call is to be pasted. Select `sample4.c` (`cvi\tutorial\sample4.c`) from the dialog box and press <Enter> or click on **OK**.

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

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

Your program should match the source code shown in the following illustration.



9. 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>.

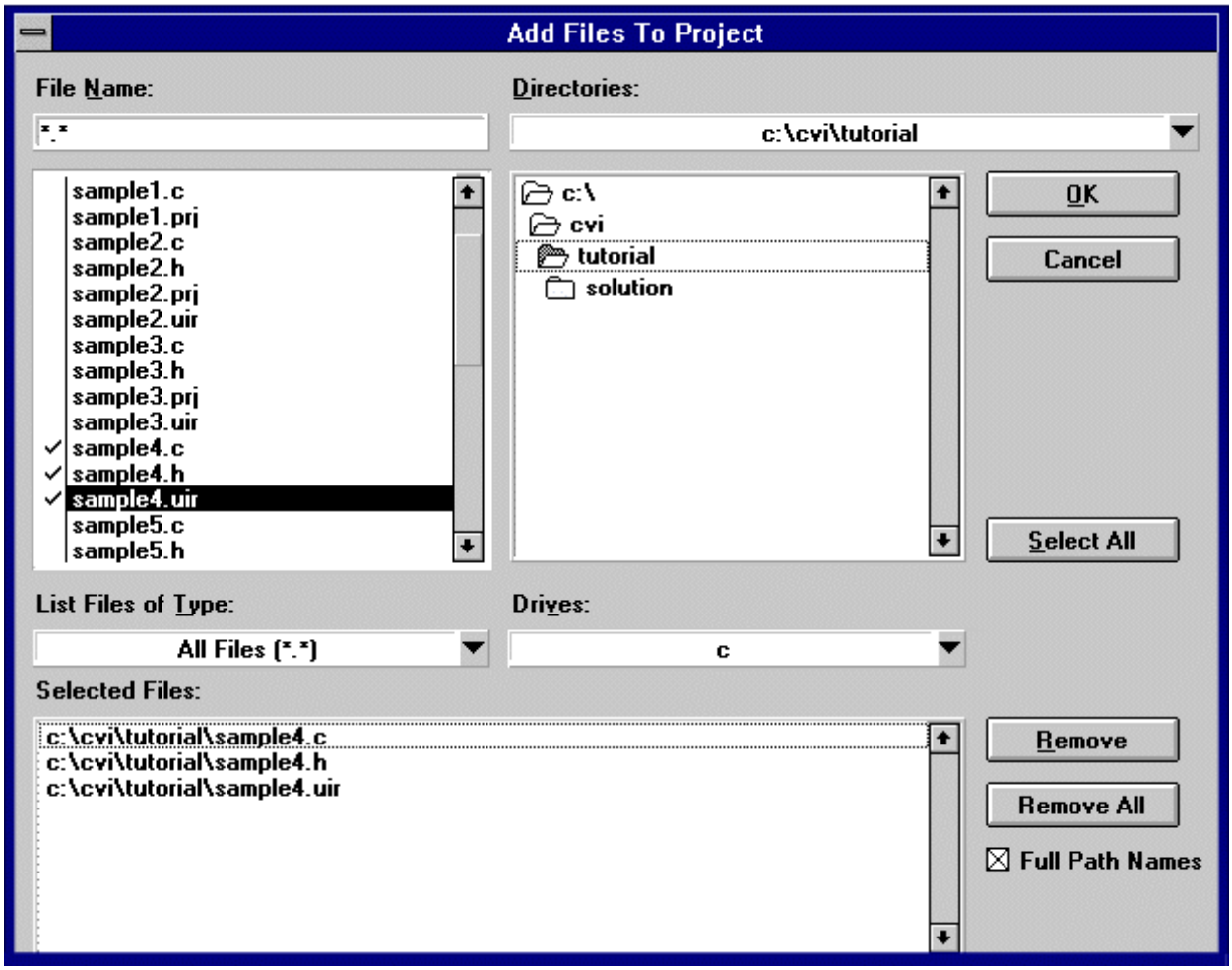
Constructing the Project

In the first tutorial session (Chapter 2), 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 will display a dialog box listing 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 the following illustration.)



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.

Chapter 7
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 being generated. 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 created 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.

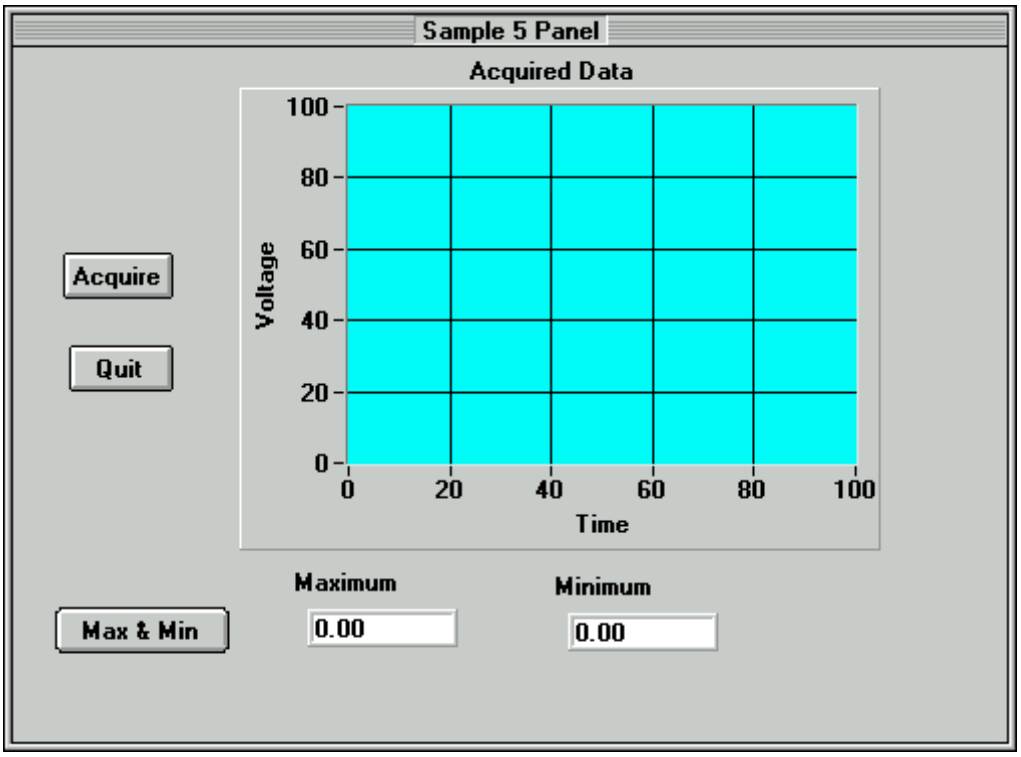
1. Add a command button control to the user interface.
2. Add two numeric readout controls to the user interface.
3. Set up a callback function in the source code file to be triggered by the added command button.
4. Generate source code from the Analysis Library to find the maximum and minimum values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

Since 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 where you left off with 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.
2. Without closing the `sample5.c` source code, return to the Project window and double-click on the `sample5.uir` filename in the Project window to open the User Interface Editor. Your goal is to modify the `.uir` to match the user interface shown in the following illustration.



3. Select **Command Button** from the **Create** menu and choose a command button from the control palette.

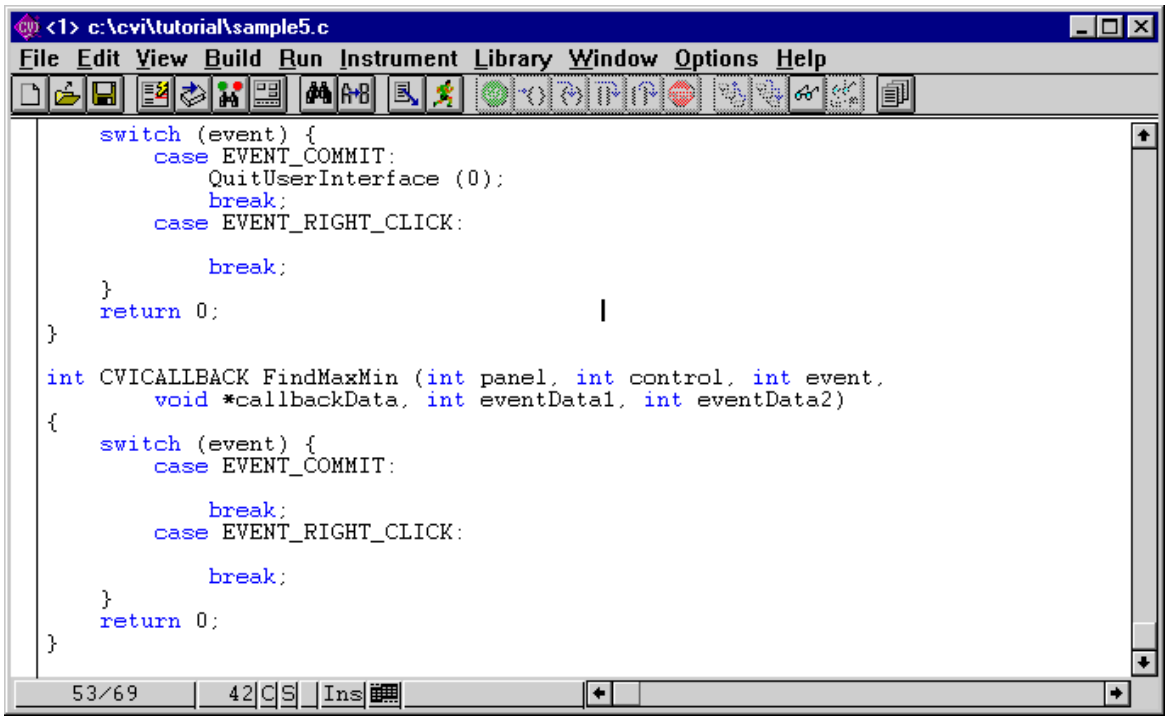
4. Double-click on the **Command** button to bring up the dialog box. Enter the following information into the dialog box.

   Constant Name: `MAXMIN`

   CallbackFunction: `FindMaxMin`

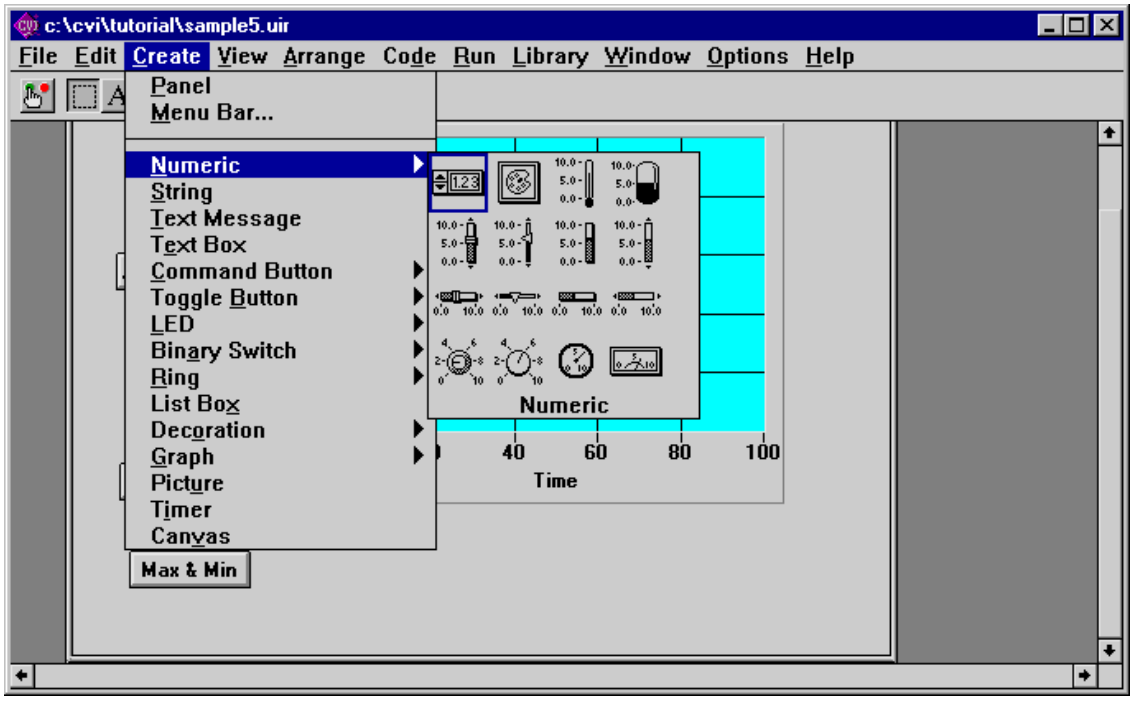
   Label: `Max & Min`

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.



The lightning-bolt cursor appears while CodeBuilder generates code into the `sample5.c` source file as shown above. 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 the following illustration.



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`

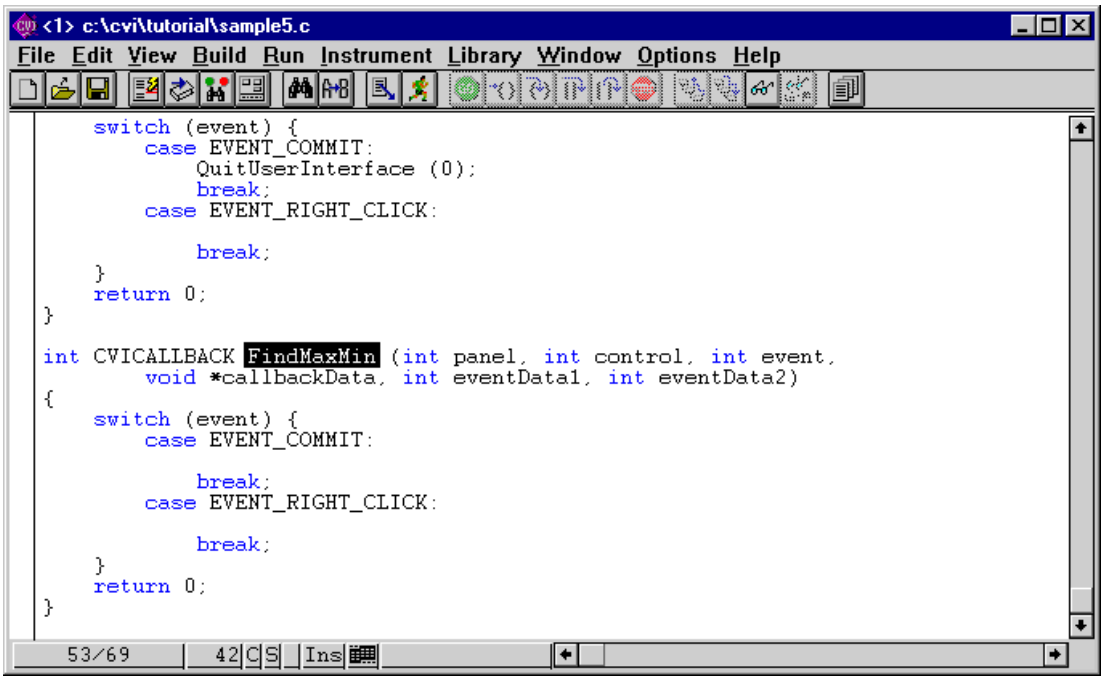
9. Position the two new controls on the user interface to match those in the illustration of the Sample 5 Panel shown previously.

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 follows.



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 whenever 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 **Analysis>Array Operations>1D Operations>1D Maximum & Minimum** from the **Library** menu.

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

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

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

5. Before inserting 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 will insert a line of code to declare the `max` variable within the `FindMaxMin` callback function. Press the **OK** button to continue.

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

7. Repeat the **Declare Variable** steps for the **Maximum Index**, **Minimum Value**, and **Minimum Index** controls on the `MaxMin1D` function panel.

8. Insert the `MaxMin1D` function call into your source code by selecting **Insert Function Call** from the **Code** menu. Close the `MaxMin1D` function panel. You should see the `MaxMin1D` function inside the `EVENT_COMMIT` case statement in the function `FindMaxMin` of your source code.

9. Display the `SetCtrlVal` function panel by selecting **User Interface**>**Controls/Graphs/Strip Charts**>**General Functions**>**Set Control Value** from the **Library** menu.

10. The `SetCtrlVal` function sets the value of a control on your user interface. Enter the following information into the function panel controls to display the maximum value at the array in the Maximum numeric display.

    Panel Handle: `panelHandle`

    Control ID: `PANEL_MAX`

    Value: `max`

11. Insert the `SetCtrlVal` function call into your source code by selecting **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.

12. Now display the `SetCtrlVal` function panel again by selecting **User Interface**>**Controls/Graphs/Strip Charts**>**General Functions**>**Set Control Value** from the **Library** menu.

13. 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 at the array in the Minimum numeric display: Close the `SetCtrlVal` function panel. The new code will appear after the previously inserted `SetCtrlVal` function call.

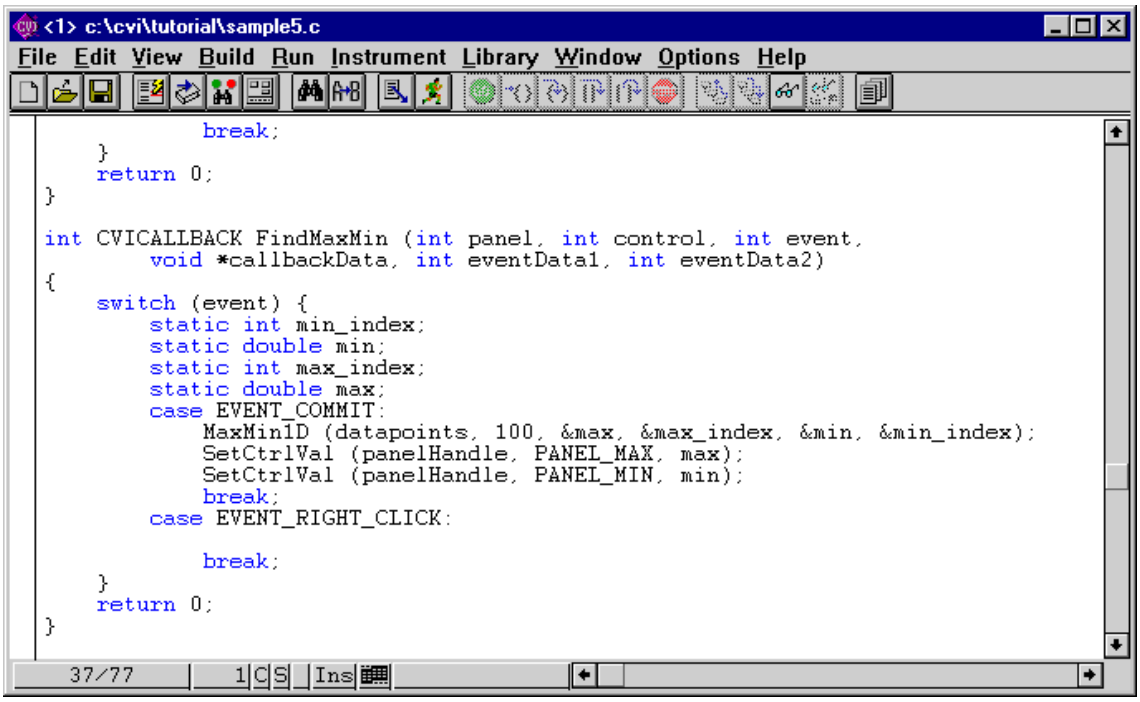
    Panel Handle: `panelHandle`

    Control ID: `PANEL_MIN`

    Value: `min`

14. Insert the `SetCtrlVal` function call into your source code by selecting **Insert Function Call** from the **Code** menu. Close the `SetCtrlVal` function panel.

15. Your source code should match the code shown in the following illustration.



Running the Program

You have now successfully written your own callback function. During program execution, the `FindMaxMin` function will be called whenever 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 additional 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.

Chapter 8
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:

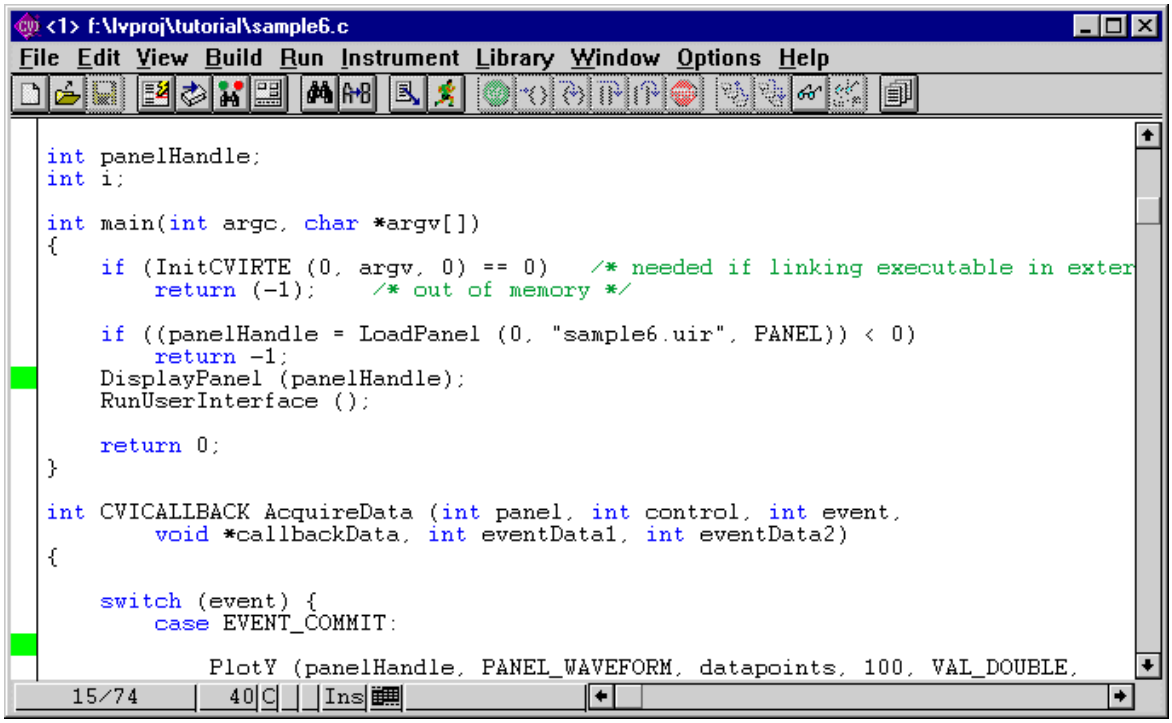
1. Select **Open** from the **File** menu. Choose `Project (*.prj)` as the file type.
2. Select `sample6.prj` from the dialog box.
3. Close all windows except for the Project window.

Loading the Instrument Driver

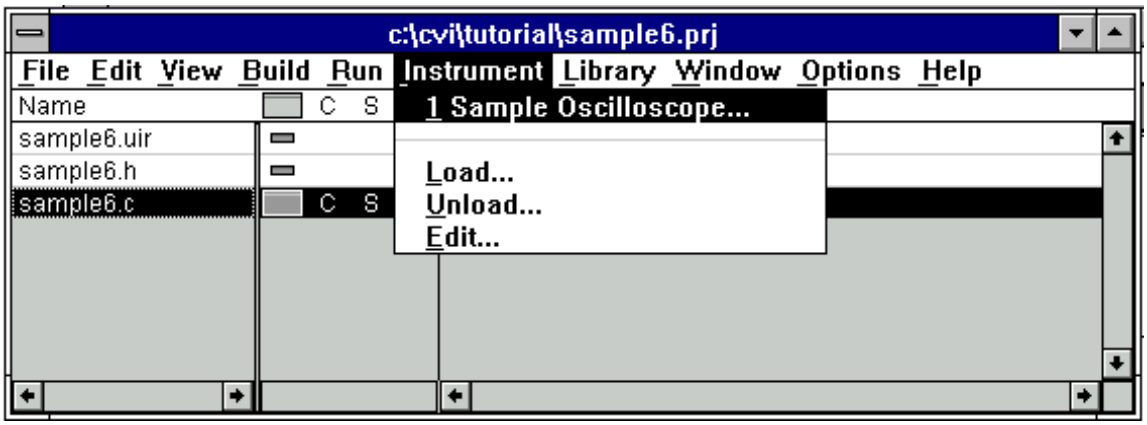
An instrument driver consists of several files that reside on disk. Use the **Instrument** menu to load these files for use in LabWindows/CVI. Perform the following steps to load the sample instrument driver:

1. Select **Load** from the **Instrument** menu in the Project window. The Load Instrument dialog box appears.
2. Press <Tab> to move the highlighting to the large list box containing directory and file names. (Click anywhere inside the list box if you are using a mouse.)
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 the following illustration. 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.

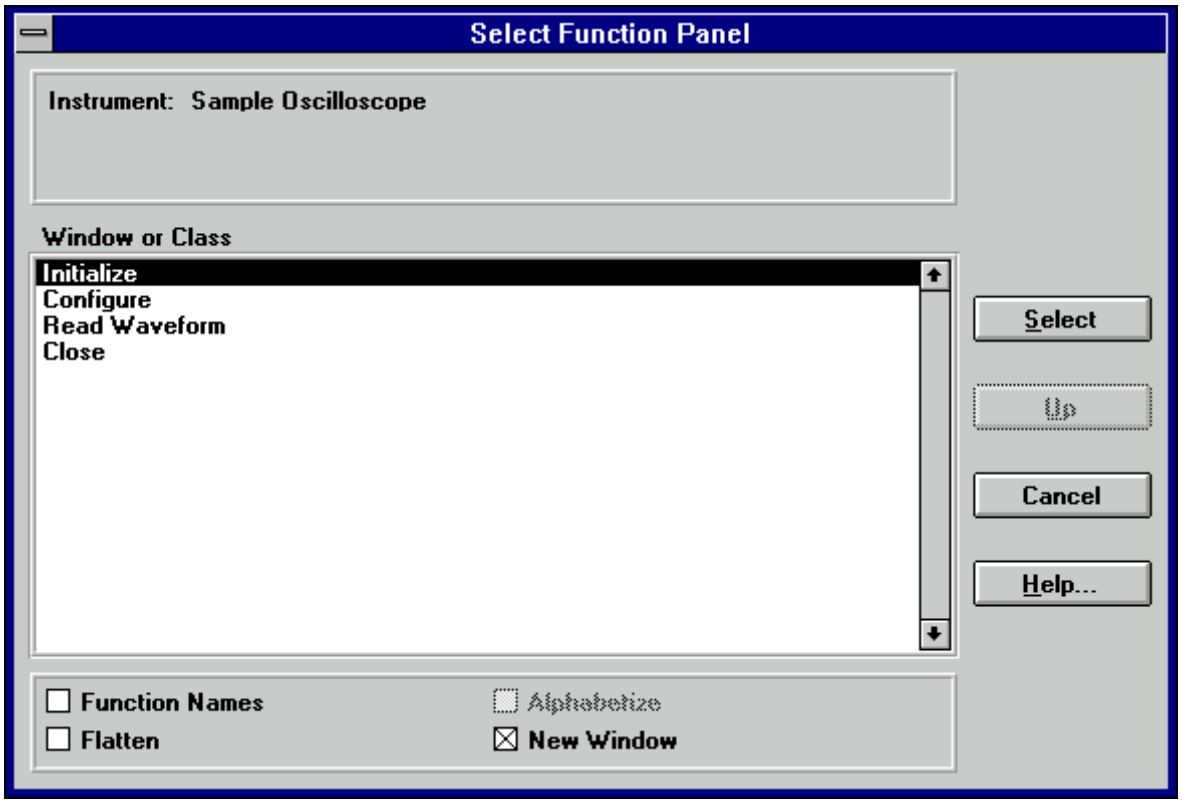


To verify that the Scope instrument driver was loaded, display the **Instrument** menu. The **Sample Oscilloscope** item should appear on the menu as shown in the following illustration.



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 via menus, dialog boxes, and function panels. Select the **Sample Oscilloscope** item from the **Instrument** menu. The dialog box shown in the following illustration appears.



This module contains the following functions.

- Initialize
- Configure
- Read Waveform
- Close

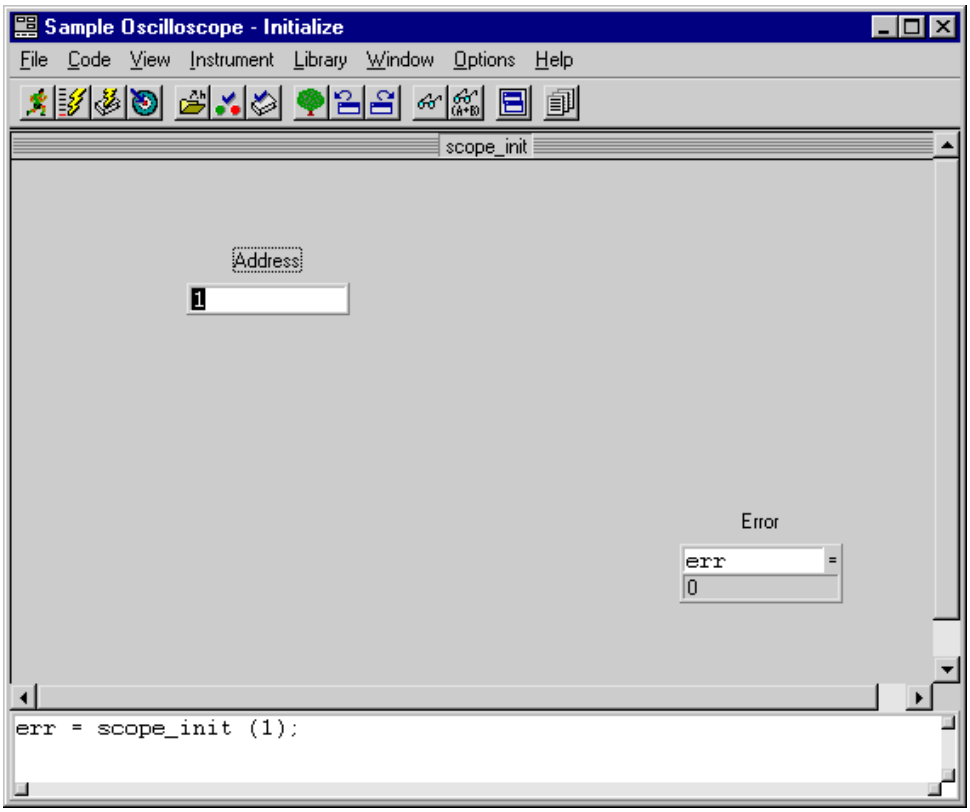
You will use these functions to acquire a waveform in the sample program you develop throughout this tutorial. Leave this dialog box open. You will select a function after you read more about how to execute function panels before inserting code into your program.

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 the following illustration appears.



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 you get a dialog box asking whether you want to save changes before running. This dialog box will come up 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.

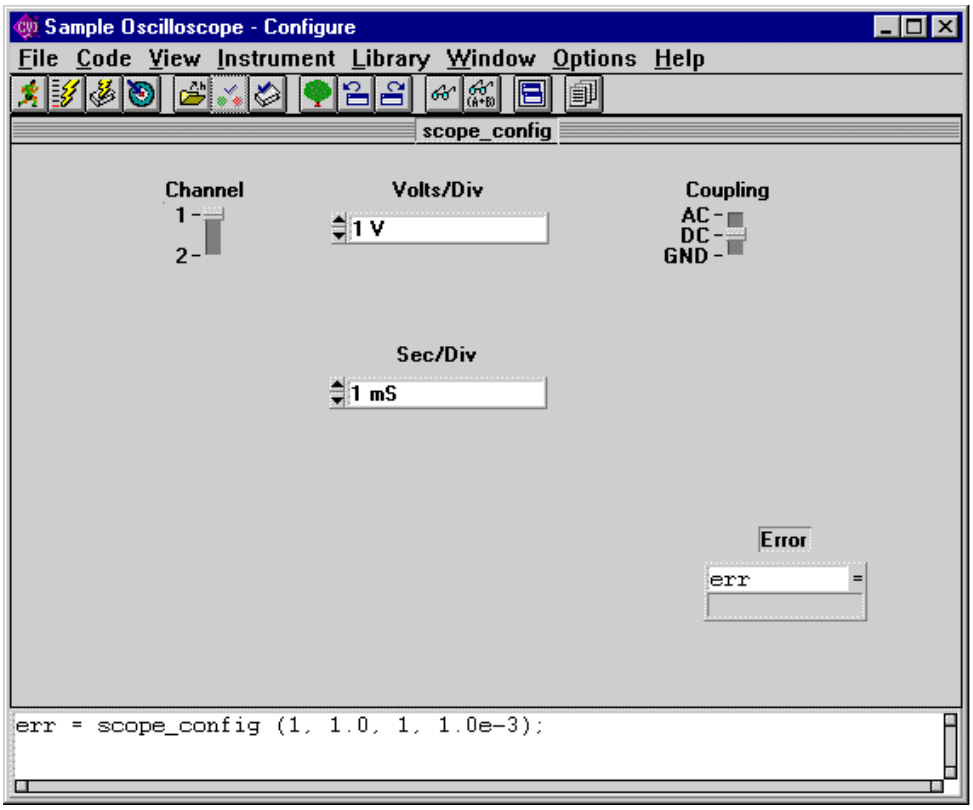
2. You may have to reset the target file for the inserted code. When you try to insert the code, the dialog box will prompt you for the target file if you have not set it. Select `sample6.c` from the dialog box.

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 the following illustration appears on your screen.



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 the settings are configured will affect 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.

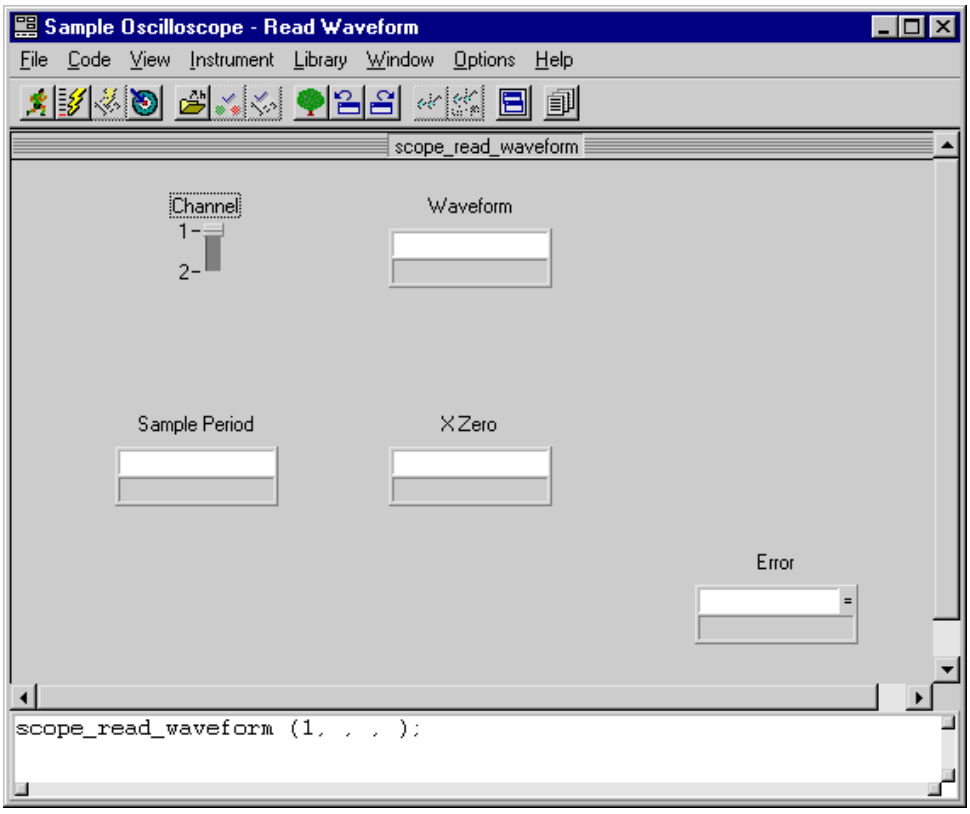
3. Select **Insert Function Call** from the **Code** menu to copy the generated code to the source window. Close the function panel.

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 directly usable by your program. 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 the following illustration.

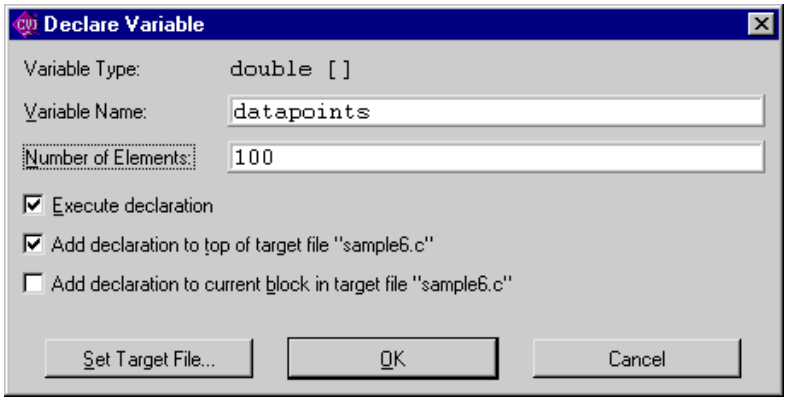


Set the Channel control to the channel you want to read. Channel 1 is a sinewave and Channel 2 is random data.

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 the following illustration.



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> to highlight the Sample Period control. (With a mouse, 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 your mouse input 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>     /* needed if linking executable in external compiler;
harmless otherwise */
#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)   /* needed if linking executable in
external compiler; harmless otherwise */
        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` ***may differ from that 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.

Chapter 9
Additional Exercises

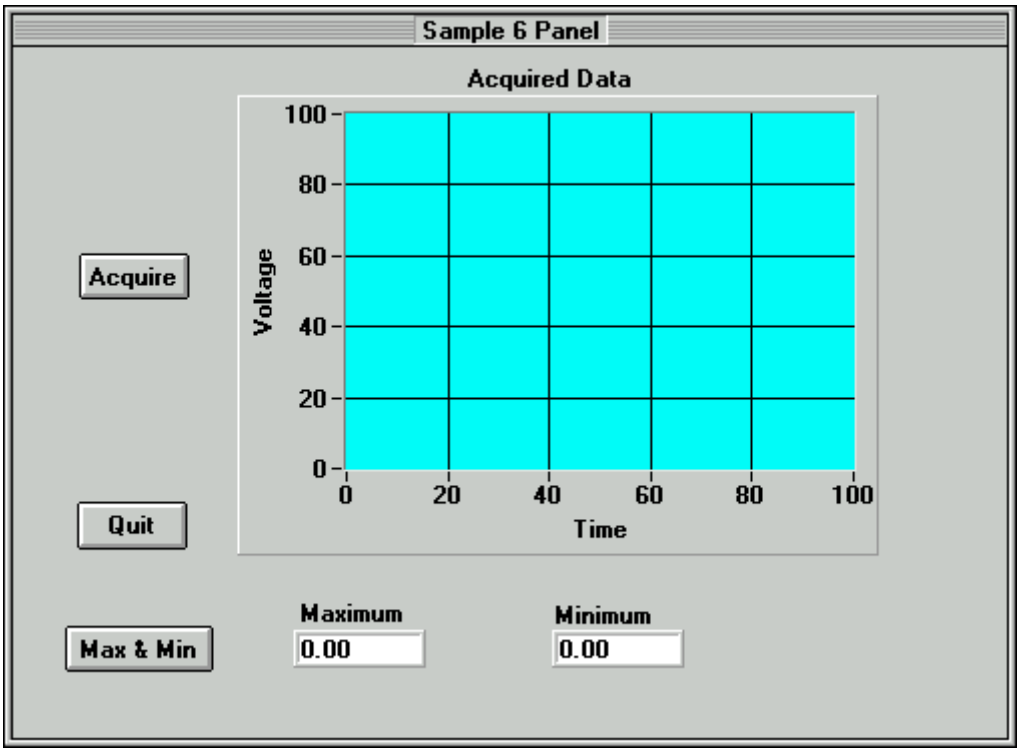
This is the last chapter of the tutorial sessions. By working the exercises in this chapter, you will learn more about the concepts you have been using throughout this tutorial. Each exercise builds on the code that you developed in the previous exercise. Exercise 1 starts by building on the final sample program that you completed in the Chapter 8 tutorial session. You are given 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 the `SOLUTIONS` subdirectory of the `TUTORIAL` directory. 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 the following illustration.

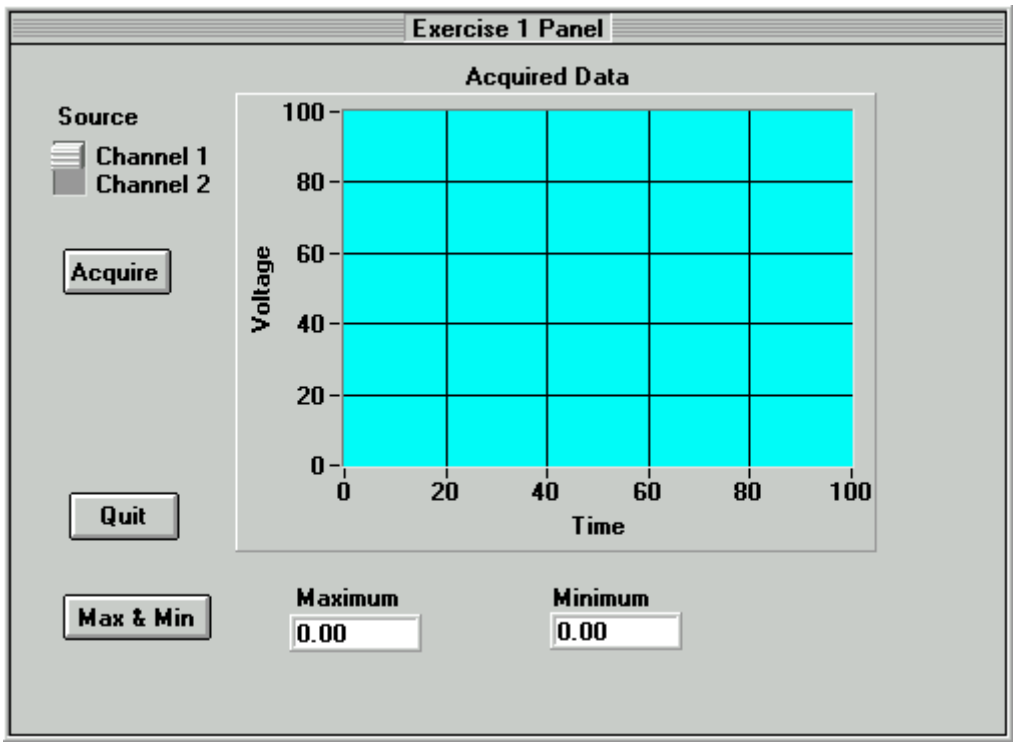


Exercise 1: Add a Channel Control

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 the following illustration, and modify the source code to properly acquire the correct channel.



Hints:

1. Use a binary switch for the channel select control.

2. Use the `GetCtrlVal` function in the `AcquireData` callback function to find out which channel the user has selected.

3. Use the value received 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 building 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 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 has clicked on the **Acquire** button. Use the `SetCtrlAttribute` function to enable the **Max & Min** button when a user has clicked on the **Acquire** button.

Hints:

1. Start by disabling (dimming) the **Max & Min** command button in the User Interface Editor.
2. Use the `SetCtrlAttribute` function from the User Interface Library to enable the **Max & Min** button.
3. The attribute that you will be setting 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 archive 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:

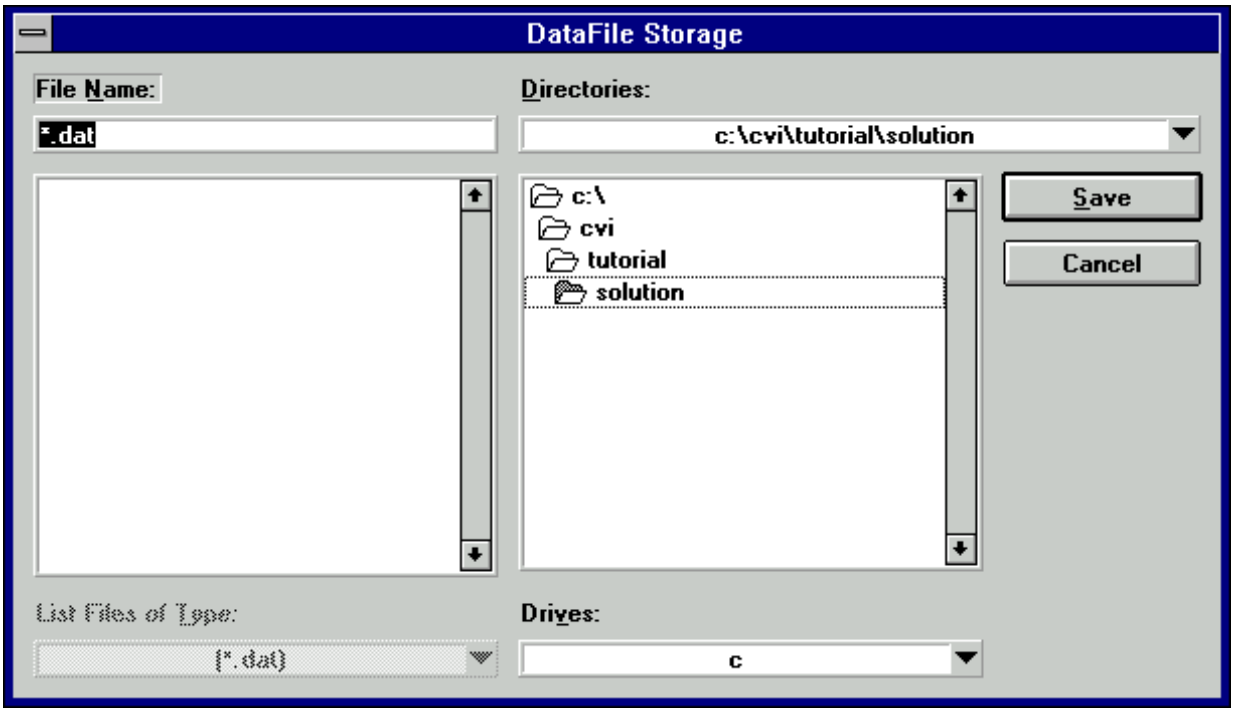
1. Remember that you must first open a file before you can write to it.
2. Open the file as a text file, so you can view the contents in any text editor later.
3. Open the file with the Create/Open flag, and not the Append flag, so that the file will be overwritten each time.
4. Use the `fprintf` function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

Solution: `EXER3.PRJ`

Exercise 4: Pop-up Panels

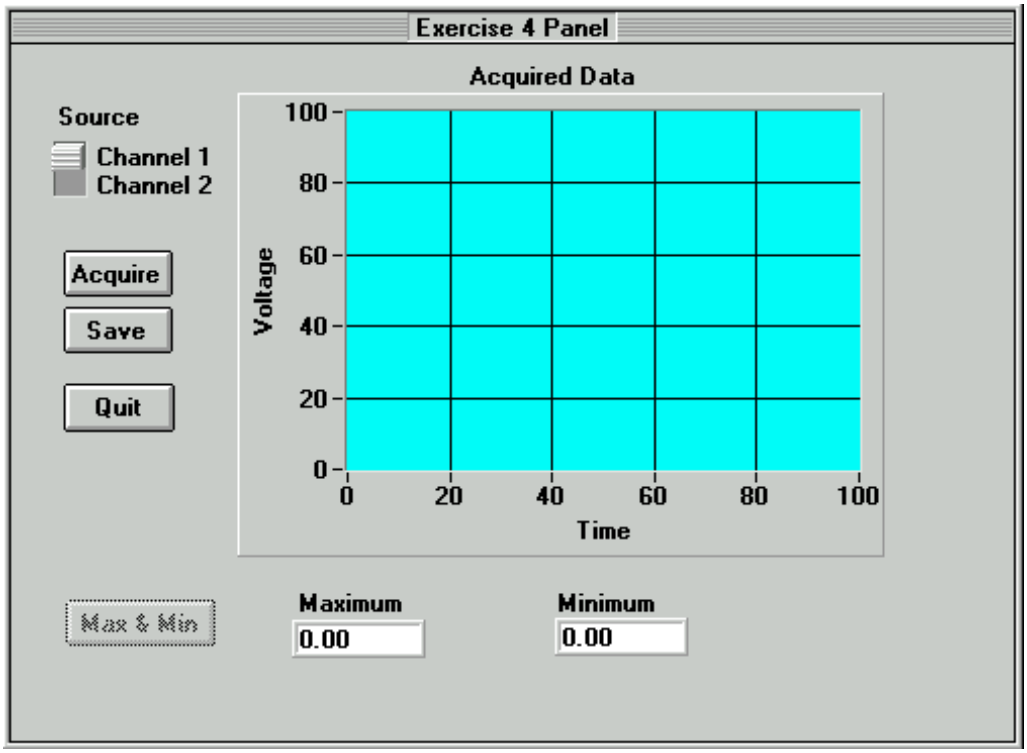
The User Interface Library has a set of predefined panels called Pop-up Panels. Pop-up Panels provide a quick and easy way to display information on the screen without developing a complete `.uir` file. In the Chapter 5 tutorial, you used a pop-up panel to display the random number array on a graph (`YGraphPopup`). 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 to prompt the user to select or input a file name.



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 the following illustration.



Hints:

1. When you create the **Save** button, assign a callback function to it.

2. You must move the source code that you developed in Exercise 3 for writing the array to disk into the callback function.

3. 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-mouse button click or a right-mouse button click. Or, an event could be a left-mouse button 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 is 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 is to process the `EVENT_LEFT_CLICK` event, and the third time is 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 different from the person who developed the program. Although using a program with a GUI can be intuitive, it may 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:

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

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.

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 every 1 seconds.

Hints:

1. Set the timer interval to 1.
2. Use CodeBuilder to generate the shell for your Timer Control callback function.
3. Use `SetCtrlVal` to display the random number on the thermometer.

Solution: `EXER6.PRJ`

Chapter 10
Getting Started with GPIB and VXI Instrument Control

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

Getting Started with Your GPIB Controller

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

Introduction to GPIB

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

In 1987, 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 and IEEE 488.2 specifications for controllers. The National Instruments IEEE 488.2 compatible TNT4882C and NAT4882 GPIB controller ASICs continue to improve and advance GPIB communication.

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 & 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)
SB-GPIB/TNT & SB-GPIB (Plug-in SBus interface)
GPIB-SCSI-A (External SCSI-GPIB Controller)
GPIB-ENET (External Ethernet TCP/IP-GPIB Controller)

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

Configuring Your GPIB Driver Software

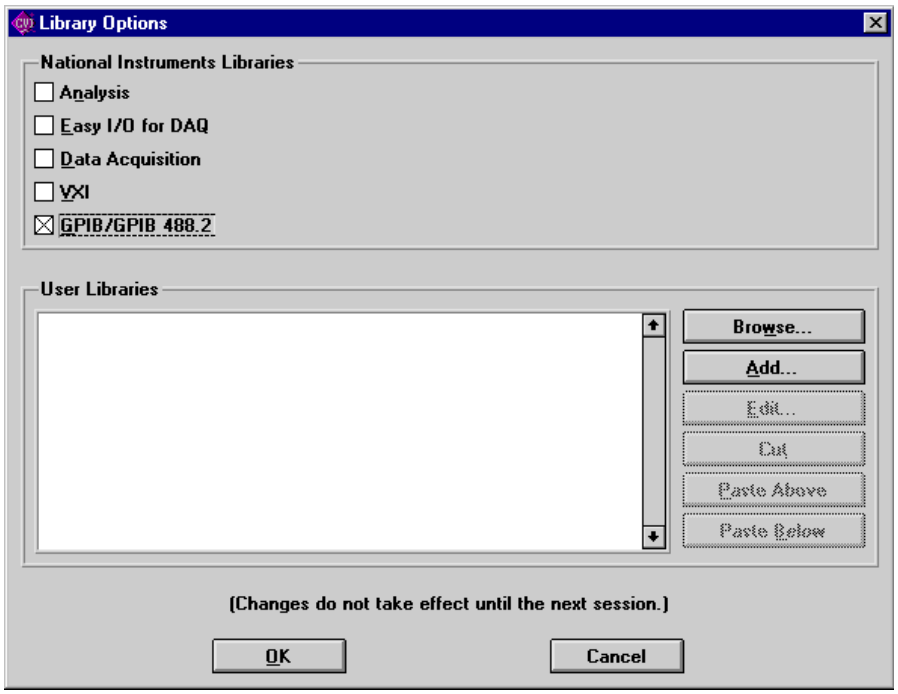
NI-488.2 is more than just a library of routines for controlling GPIB instruments. NI-488.2 includes a number of software utilities for testing and configuring the operation of your controller. Some of these utilities include the following.

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

These and other utilities are described in the documentation that you received with your GPIB hardware. For Windows users, refer to the *NI-488.2 User Manual for Windows.* Sun users can find this information in the *NI-488.2M Software Reference Manual.*

Configuring LabWindows/CVI for GPIB

LabWindows/CVI uses the NI-488.2 DLL for Windows and NI-488.2M device driver for Sun Solaris 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 the following illustration.



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). On the Sun, 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 *NI-488.2 Function Reference Manual* for Windows users and in the *NI-488.2M Software Reference Manual* for Sun users.

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 VXI Development System (VXS) contains software for controlling VXI instruments for any of the methods mentioned above. The VXS contains the following items.

- The LabWindows/CVI full development system
- Libraries for controlling National Instruments VXI controllers from LabWindows/CVI
- The complete LabWindows/CVI VXI Instrument Library for controlling instruments from a wide variety of vendors

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 VXS to control your VXI instruments.

Installing and Configuring Your VXI Hardware

The LabWindows/CVI VXS works with the following VXI controllers.

LabWindows/CVI for Windows

VXIpc Model 500 and 850 Series
VXIpc-486 Model 240
AT-MXI and AT-MXI-2 controllers
PCI-MXI-2 controllers
MC-MXI controllers
GPIB-VXI/C

LabWindows/CVI for Sun

SB-MXI
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

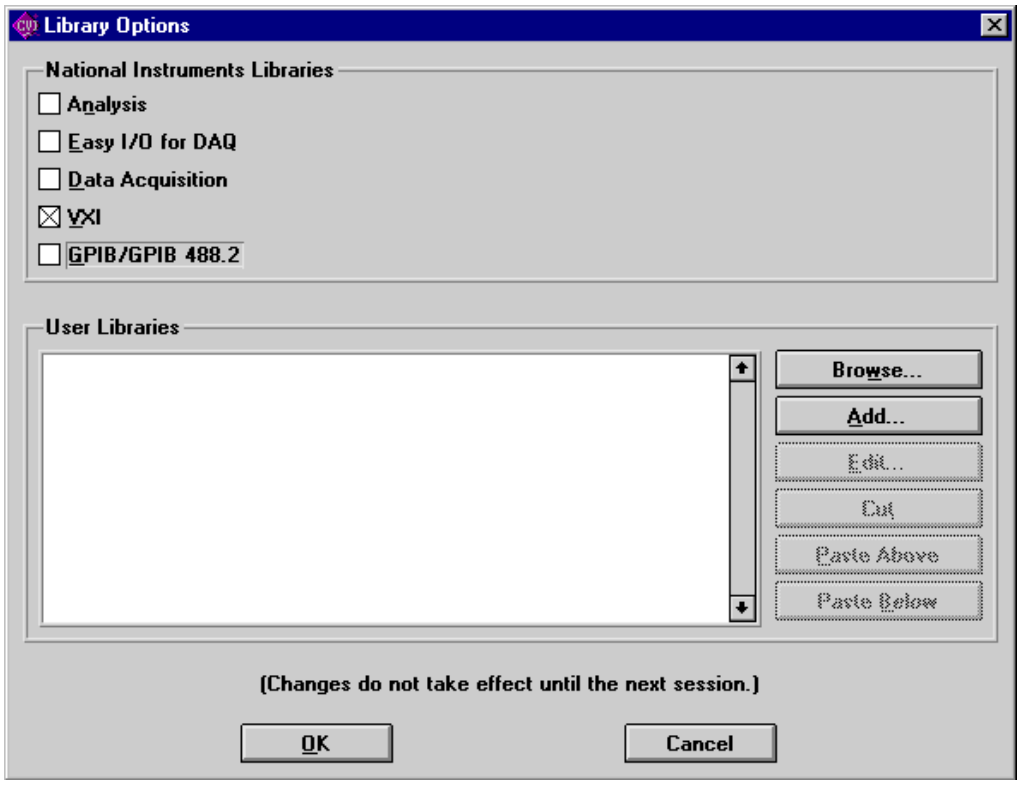
NI-VXI is more than just a library of routines for controlling your VXIbus instruments. NI-VXI, like NI-488.2, contains configuration and troubleshooting utility software for your VXIbus system. Some of these utilities include the following.

- `VXIINIT`—An initialization program for board interrupts, shared RAM, VXI register configurations, and bus configurations for MXI controllers.
- `VXITEDIT`—A text-based editor for setting or changing your VXIbus system configuration.
- `RESMAN`—The National Instruments multimainframe Resource Manager.
- `VIC`—An interactive control program that executes VXI functions that you enter from the keyboard. (Available as DOS executable for Windows users only)

Refer to the Getting Started manual that you received with your VXI controller to use these utilities for configuring your system. For more detailed information on these programs, refer to the *NI-VXI Text Utilities Reference Manual*.

Configuring LabWindows/CVI for VXI

LabWindows/CVI uses the NI-VXI DLL for Windows and NI-VXI device driver for Sun Solaris 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 the following illustration.



Developing Your Application

The LabWindows/CVI VXS 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. On the Sun, this library is a shared object. While the function panels in LabWindows/CVI provide online help information for using these functions, detailed function descriptions for the VXI Library functions can be found in the *NI-VXI Software Reference Manual for C*.

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 which 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. Simply 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 on using instrument drivers in your application, refer to Chapter 8, *Using an Instrument Driver*, in this manual. If you plan on developing an instrument driver yourself, refer to the *LabWindows/CVI Instrument Driver Developers Guide* to learn how to use the development tools for creating function trees, function panels, and instrument control source code for a driver.

Chapter 11
Getting Started with Data Acquisition

This chapter is a quick reference for installing and configuring National Instruments plug-in data acquisition (DAQ) devices for use with LabWindows/CVI for Windows. The information included in this chapter is presented in more detail in the documentation that your 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 onboard.

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 will automatically load 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 plugging in your DAQ device into your computer.

Configure Your Jumpers and DIP Switches

A DAQ device may 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, analog input polarity, and input mode (single-ended or differential).

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

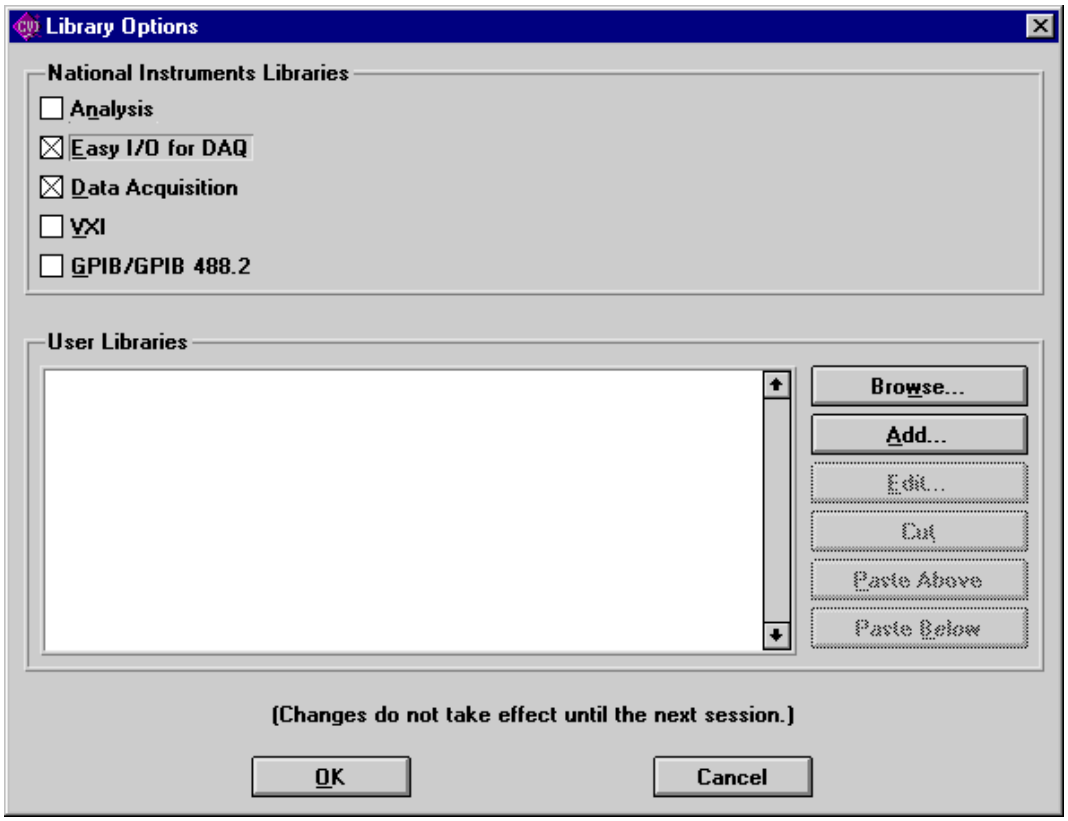
Software Installation

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 the following illustration.



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. There are two levels of simple testing 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 `AISampleChannel` function acquires a single voltage from a single analog input channel. You can connect a known voltage source to your DAQ device and verify that the analog input reading changes accordingly.

– AO Sample Channel—The `AOSampleChannel` function applies a specified voltage to a single analog output channel.

For more information on how to execute LabWindows/CVI function panels, 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, *An 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 installed 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`.

Note: ***If you previously used the SCXI Analog Input instrument driver, you can replace those functions in your program with functions from this library. Then, you will no longer need to use the SCXI Analog Input instrument driver.***

Data Acquisition Library Sample Programs

The following are a few of the example programs installed into the `cvi\samples\daq` directory by your NI-DAQ for Windows software.

- `DAQ_OP` synchronously acquires data from one analog input channel and displays the data on a strip chart. The `DAQ_OP` function will not return until all the data requested has been acquired.
- `DAQSTART` asynchronously acquires data from one analog input channel and displays the data on a graph.

- `DBLBUFFR` asynchronously acquires data in a double-buffered acquisition operation from one analog input channel and continuously displays the data on a graph. The technique illustrated in this example is ideal for continuous data acquisition, analysis, and streaming to disk. Remember that plotting data on a graph can slow the rate at which data streams to disk.
- `DIGITAL` updates the digital I/O lines from a graphical user interface.
- `EVENTCNT` counts the pulses that are input to a counter/timer and updates a numeric indicator on the user interface.
- `FREQMEAS` measures the frequency of a signal using the device counter/timer inputs.
- `PULSWDTH` measures the width of a pulse applied to the input of a counter/timer on the device.
- `WVFMDBL` generates a continuous analog output signal using a double-buffered operation.
- `WVFMOUT` generates a synchronous analog output signal.

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 which 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\toolslib\custctrl\daq_num\daqndemo.prj`.

DAQ Chart Control Instrument Driver

The DAQ Chart Control instrument driver implements strip chart controls which can automatically scan a set of analog input channels at a specified rate and update the strip chart traces. The chart can be configured 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\toolslib\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 and 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.

The following table shows the typedefs and the intrinsic types under for the different platforms.

Table 5-1. 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 or 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 information you may find helpful as you read this chapter.

- *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*

Chapter 12
Converting LabWindows for DOS Applications

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

Conversion Tools

You may 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 they can be used in LabWindows/CVI. See 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 they can be used in LabWindows/CVI. See the section, *Converting User Interface Resource Files*, for details about converting `.uir` files.
- The function panel (`.fp`) file translator converts LabWindows for DOS `.fp` files so they can be used in LabWindows/CVI. See the section, *Converting Instrument Drivers*, 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 via 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. See 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.

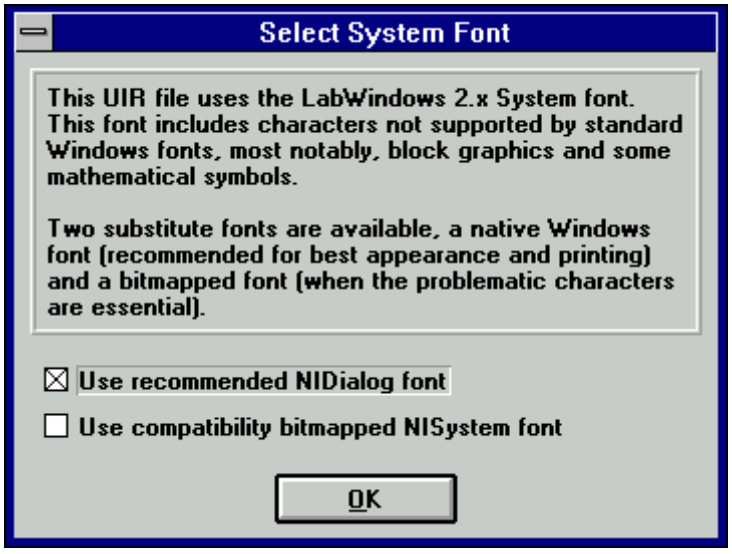
The Utility Library

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 they can be used 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. The file will be automatically converted and you will be given 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 the following illustration is displayed.



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 need to make cosmetic changes manually. Once you are satisfied with the appearance of the `.uir` file, save it so that it can be used by your LabWindows/CVI program.

Converting Source Code

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

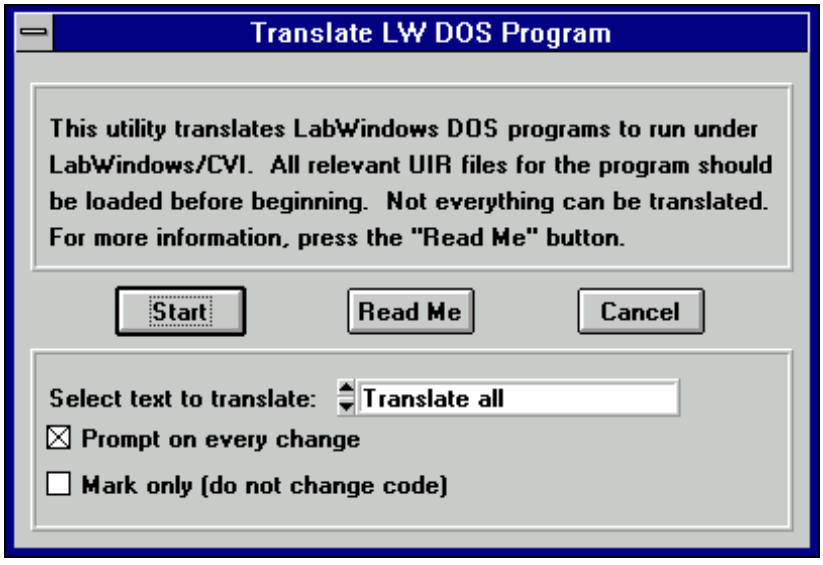
To convert your C source code follow these steps:

1. Open any `.uir` files that your program uses. (See the section, *Converting User Interface Resource Files*, in this chapter if they have not yet been converted.)

2. Backup 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 DOS LW Program** from the **Options** menu. This will invoke the dialog box shown in the following illustration.



- **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. Once you have configured the Translate DOS LW 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. The array should be declared as `short` instead, and any type specifiers that refer to it should have the `[b2]` modifier when passed as an argument to a Formatting and I/O Library function.

8. You will need to modify any `int` variables that are used in a way that requires them to be two-byte integers.

   For example, if an int argument is passed by address to a function in the Formatting and I/O Library (a `Scan` source or a `Scan/Fmt` target) and 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 a `short` argument is passed by address and 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 below.

   In LabWindows for DOS, the `[o]` modifier is used 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.

   Since LabWindows/CVI supports both Intel (PC) and Motorola (SPARCstation) architectures, the `[o]` modifier in LabWindows/CVI is used 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 will reverse 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, *The 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, it is recommended that you acquire new LabWindows/CVI instrument drivers from National Instruments.

This section describes the procedure for converting your LabWindows for DOS instrument driver files so that they can be used 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 be porting 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, selection 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 may compile it using LabWindows/CVI or a compatible external compiler. See 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 are not directly supported by LabWindows/CVI. Under Windows, these modules will have to be rewritten in DLL form using the Microsoft DDK. Under Solaris, hardware-specific modules will have to be rewritten using the host system library.***

Appendix
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

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 or (800) 327-3077
  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: 1 48 65 15 59
  Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access FaxBack from a touch-tone telephone at the following number: (512) 418-1111.

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.

E-Mail Support (currently U.S. only)

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

GPIB:	`gpib.support@natinst.com`
DAQ:	`daq.support@natinst.com`
VXI:	`vxi.support@natinst.com`
LabVIEW:	`lv.support@natinst.com`
LabWindows:	`lw.support@natinst.com`
HiQ:	`hiq.support@natinst.com`
Lookout:	`lookout.support@natinst.com`
VISA:	`visa.support@natinst.com`

Fax and Telephone 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.

	Telephone	Fax
Australia	03 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	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
U.K.	01635 523545	01635 523154

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: Windows 3.1, Windows for Workgroups 3.11, Windows NT 3.1, Windows NT 3.5, Windows 95, other (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 will reproduce the problem _____

_____

_____

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. When you complete this form accurately before contacting National Instruments for technical support, our applications engineers can answer your questions more efficiently.

National Instruments Products

Data Acquisition Hardware Revision _____

Interrupt Level of Hardware _____

DMA Channels of Hardware _____

Base I/O Address of Hardware _____

NI-DAQ, LabVIEW, or
LabWindows Version _____

Other Products

Computer Make and Model _____

Microprocessor _____

Clock Frequency _____

Type of Video Board Installed _____

Operating System _____

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: **July 1996**

Part Number: **320680C-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 _____

_____

Fax (_____)_____ Phone (_____)_____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 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 upon 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 are indicated in dialog boxes with an outline.
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

.FP file	A file containing 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 Library	A LabWindows/CVI library that contains instrument control functions.
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 via menus, dialog boxes, and function panels.
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 containing code stored in LabWindows for DOS binary format, a format translatable into C or BASIC source code by the LabWindows for DOS environment.
`.LWI` file	A file containing instrument include statements in LabWindows for DOS binary format.
list box	A dialog box item that displays a list of possible choices for completing a command in the dialog box.

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 to which 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. Scroll bars can be used with a mouse to move about in the window.
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 provides a means of executing a command without accessing a menu in the command bar.
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. This window is designated by the file extension `.c`.
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 text is entered 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 Library	Includes a set of functions for controlling the interface programmatically, as well as a resource editor for defining the user interface components.
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.

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

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Y