LabWindows™/CVI™

Getting Started with LabWindows/CVI

August 2015
373552K-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information
ni.com

Worldwide Offices
Visit ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

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

For further support information, refer to the *NI Services* appendix. To comment on National Instruments documentation, refer to the National Instruments website at ni.com/info and enter the Info Code `feedback`.

© 1994–2015 National Instruments. All rights reserved.

Legal Information

Limited Warranty

This document is provided 'as is' and is subject to being changed, without notice, in future editions. For the latest version, refer to `ni.com/manuals`. NI reviews this document carefully for technical accuracy; however, NI MAKES NO EXPRESS OR IMPLIED WARRANTIES AS TO THE ACCURACY OF THE INFORMATION CONTAINED HEREIN AND SHALL NOT BE LIABLE FOR ANY ERRORS.

NI warrants that its hardware products will be free of defects in materials and workmanship that cause the product to fail to substantially conform to the applicable NI published specifications for one (1) year from the date of invoice.

For a period of ninety (90) days from the date of invoice, NI warrants that (i) its software products will perform substantially in accordance with the applicable documentation provided with the software and (ii) the software media will be free from defects in materials and workmanship.

If NI receives notice of a defect or non-conformance during the applicable warranty period, NI will, in its discretion: (i) repair or replace the affected product, or (ii) refund the fees paid for the affected product. Repaired or replaced Hardware will be warranted for the remainder of the original warranty period or ninety (90) days, whichever is longer. If NI elects to repair or replace the product, NI may use new or refurbished parts or products that are equivalent to new in performance and reliability and are at least functionally equivalent to the original part or product.

You must obtain an RMA number from NI before returning any product to NI. NI reserves the right to charge a fee for examining and testing Hardware not covered by the Limited Warranty.

This Limited Warranty does not apply if the defect of the product resulted from improper or inadequate maintenance, installation, repair, or calibration (performed by a party other than NI); unauthorized modification; improper environment; use of an improper hardware or software key; improper use or operation outside of the specification for the product; improper voltages; accident, abuse, or neglect; or a hazard such as lightning, flood, or other act of nature.

THE REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND THE CUSTOMER'S SOLE REMEDIES, AND SHALL APPLY EVEN IF SUCH REMEDIES FAIL OF THEIR ESSENTIAL PURPOSE.

EXCEPT AS EXPRESSLY SET FORTH HEREIN, PRODUCTS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND AND NI DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, WITH RESPECT TO THE PRODUCTS, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT, AND ANY WARRANTIES THAT MAY ARISE FROM USAGE OF TRADE OR COURSE OF DEALING. NI DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF OR THE RESULTS OF THE USE OF THE PRODUCTS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NI DOES NOT WARRANT THAT THE OPERATION OF THE PRODUCTS WILL BE UNINTERRUPTED OR ERROR FREE.

In the event that you and NI have a separate signed written agreement with warranty terms covering the products, then the warranty terms in the separate agreement shall control.

Copyright

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

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

End-User License Agreements and Third-Party Legal Notices

You can find end-user license agreements (EULAs) and third-party legal notices in the following locations:

- Notices are located in the `<National Instruments>\_Legal Information` and `<National Instruments>` directories.
- EULAs are located in the `<National Instruments>\Shared\MDF\Legal\license` directory.
- Review `<National Instruments>\_Legal Information.txt` for information on including legal information in installers built with NI products.

U.S. Government Restricted Rights

If you are an agency, department, or other entity of the United States Government ("Government"), the use, duplication, reproduction, release, modification, disclosure or transfer of the technical data included in this manual is governed by the Restricted Rights provisions under Federal Acquisition Regulation 52.227-14 for civilian agencies and Defense Federal Acquisition Regulation Supplement Section 252.227-7014 and 252.227-7015 for military agencies.

Trademarks

Refer to the *NI Trademarks and Logo Guidelines* at `ni.com/trademarks` for more information on National Instruments trademarks.

ARM, Keil, and µVision are trademarks or registered of ARM Ltd or its subsidiaries.

LEGO, the LEGO logo, WEDO, and MINDSTORMS are trademarks of the LEGO Group.

TETRIX by Pitsco is a trademark of Pitsco, Inc.

FIELDBUS FOUNDATION™ and FOUNDATION™ are trademarks of the Fieldbus Foundation.

EtherCAT® is a registered trademark of and licensed by Beckhoff Automation GmbH.

CANopen® is a registered Community Trademark of CAN in Automation e.V.

DeviceNet™ and EtherNet/IP™ are trademarks of ODVA.

Go!, SensorDAQ, and Vernier are registered trademarks of Vernier Software & Technology. Vernier Software & Technology and `vernier.com` are trademarks or trade dress.

Xilinx is the registered trademark of Xilinx, Inc.

Taptite and Trilobular are registered trademarks of Research Engineering & Manufacturing Inc.

FireWire® is the registered trademark of Apple Inc.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix®, Tek, and Tektronix, Enabling Technology are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

The ExpressCard™ word mark and logos are owned by PCMCIA and any use of such marks by National Instruments is under license.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

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

Export Compliance Information

Refer to the *Export Compliance Information* at `ni.com/legal/export-compliance` for the National Instruments global trade compliance policy and how to obtain relevant HTS codes, ECCNs, and other import/export data.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

YOU ARE ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY AND RELIABILITY OF THE PRODUCTS WHENEVER THE PRODUCTS ARE INCORPORATED IN YOUR SYSTEM OR APPLICATION, INCLUDING THE APPROPRIATE DESIGN, PROCESS, AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

PRODUCTS ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING IN THE OPERATION OF NUCLEAR FACILITIES; AIRCRAFT NAVIGATION; AIR TRAFFIC CONTROL SYSTEMS; LIFE SAVING OR LIFE SUSTAINING SYSTEMS OR SUCH OTHER MEDICAL DEVICES; OR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, “HIGH-RISK USES”). FURTHER, PRUDENT STEPS MUST BE TAKEN TO PROTECT AGAINST FAILURES, INCLUDING PROVIDING BACK-UP AND SHUT-DOWN MECHANISMS. NI EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES.

Contents

About This Manual

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

Related Documentation

The following documents contain 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.
- *LabWindows/CVI Help*
- *LabWindows/CVI Release Notes*
- *IVI Driver Development Help*
- *NI-DAQmx Help*
- *DAQ Getting Started Guide*
- *DAQ Assistant Help*
- *Traditional NI-DAQ (Legacy) C Function Reference Help*
- *NI-VISA Help*
- *NI-488.2 Help*

1 Introduction to LabWindows/CVI

LabWindows/CVI is a software development environment for C programmers. LabWindows/CVI provides powerful function libraries and a comprehensive set of software tools for data acquisition, analysis, and presentation that you can use to interactively develop data acquisition and instrument control applications.

LabWindows/CVI combines the power and flexibility of ANSI C with easy-to-use tools for building virtual instrumentation systems. A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as a plug-in board, and driver software, which together perform the functions of traditional instruments. However, virtual instruments can provide more customization, scalability, and modularity than traditional instruments.

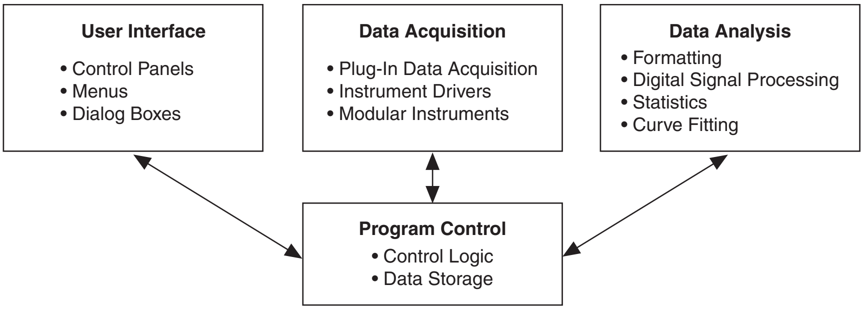
You can edit, compile, link, and debug ANSI C programs in the LabWindows/CVI development environment. Additionally, you can use compiled C object modules, DLLs, C libraries, and instrument drivers in conjunction with ANSI C source files when you develop programs.

Typical LabWindows/CVI applications include the following elements:

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

Figure 1-1 illustrates the relationship between these program elements. Program control elements receive input from the user interface, data acquisition, and data analysis elements. Each element has several sub-components.

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



When you create a virtual instrument using LabWindows/CVI and NI hardware, keep in mind the three-step process for creating virtual instruments: acquire, analyze, and present.

Acquire—Acquire your data through a hardware interface. Use the user interface you create to control data acquisition from an instrument or from a plug-in DAQ device. The user interface you create also can display the data you acquire.

Analyze—After you acquire data, you must analyze it. For example, you might want to perform formatting, scaling, signal processing, statistical analysis, and curve fitting. The LabWindows/CVI Formatting and I/O Library and Analysis Library (Base package) or Advanced Analysis Library (Full Development System) contain functions that allow you to perform these operations.

Present—Present your data in a user interface that can contain graphs, strip charts, and other controls. You also can display graphics, create pull-down menus, and prompt users for input with pop-up dialog boxes. You can use the User Interface Editor to create these items interactively, or you can use the User Interface Library to create and configure them programmatically.

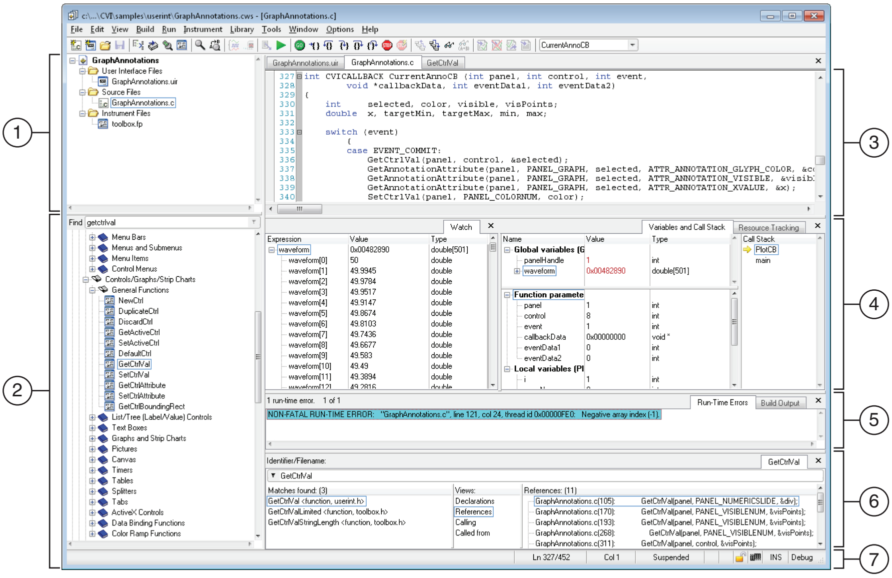
The program control portion of the program coordinates the data acquisition, data analysis, and user interface. Program control contains the control logic for managing the flow of program execution and user-defined support functions.

Use callback functions to control the flow of applications. Callback functions enable your program to execute code in response to user actions, timer ticks, and operating system events.

LabWindows/CVI Environment

The LabWindows/CVI environment is structured around the Workspace window, which is shown in the following figure:

Figure 1-2. Workspace Window



```
int CVICALLBACK CurrentAnnoCB (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int     selected, color, visible, visPoints;
    double  x, targetMin, targetMax, min, max;

    switch (event)
        {
        case EVENT_COMMIT:
            GetCtrlVal(panel, control, &selected);
            GetAnnotationAttribute(panel, PANEL_GRAPH, selected, ATTR_ANNOTATION_GLYPH_COLOR, &c
            GetAnnotationAttribute(panel, PANEL_GRAPH, selected, ATTR_ANNOTATION_VISIBLE, &visib
            GetAnnotationAttribute(panel, PANEL_GRAPH, selected, ATTR_ANNOTATION_XVALUE, &x);
            SetCtrlVal(panel, PANEL_COLORNUM, color);
```

Expression	Value	Type
waveform	0x00482890	double[501]
waveform[0]	50	double
waveform[1]	49.9945	double
waveform[2]	49.9784	double
waveform[3]	49.9517	double
waveform[4]	49.9147	double
waveform[5]	49.8674	double
waveform[6]	49.8103	double
waveform[7]	49.7436	double
waveform[8]	49.6677	double
waveform[9]	49.583	double
waveform[10]	49.49	double
waveform[11]	49.3894	double
waveform[12]	49.2816	double

Name	Value	Type
Global variables (G		
panelHandle	1	int
waveform	0x00482890	double[501]
Function paramete		
panel	1	int
control	8	int
event	1	int
callbackData	0x00000000	void *
eventData1	0	int
eventData2	0	int
Local variables (Pl		
i	1	int

Call Stack
PlotCB
main

```
1 run-time error.    1 of 1
NON-FATAL RUN-TIME ERROR:   "GraphAnnotations.c", line 121, col 24, thread id 0x00000FE0:   Negative array index (-1).
```

Identifier/Filename:
GetCtrlVal

Matches found: (3)	Views:
GetCtrlVal <function, userint.h>	Declarations
GetCtrlValLimited <function, toolbox.h>	References
GetCtrlValStringLength <function, toolbox.h>	Calling
	Called from

References: (11)	
GraphAnnotations.c(105):	GetCtrlVal(panel, PANEL_NUMERICSLIDE, &div);
GraphAnnotations.c(170):	GetCtrlVal(panel, PANEL_VISIBLENUM, &visPoints);
GraphAnnotations.c(193):	GetCtrlVal(panel, PANEL_VISIBLENUM, &visPoints);
GraphAnnotations.c(268):	GetCtrlVal(panel, PANEL_VISIBLENUM, &visPoints);
GraphAnnotations.c(311):	GetCtrlVal(panel, control, &visPoints);

Ln 327/452 | Col 1 | Suspended | INS | Debug

1	Project Tree	5	Output Region
2	Library Tree	6	Source Code Browser
3	Window Confinement Region	7	Status Bar
4	Debugging Region		

Figure 1-2 displays the following areas in the Workspace window:

- **Project Tree**—Contains the list of files in each project in the workspace. Right-click the different elements of the Project Tree to see the list of options available for files and folders.
- **Library Tree**—Contains a tree view of the functions in LabWindows/CVI libraries and instruments. You can arrange the library functions in alphabetical order, by function name or function panel title, or in a flat list instead of a hierarchical class structure. Enter a function name in the **Find** text box at the top of the Library Tree to search for a specific function within the tree.
- **Window Confinement Region**—Contains open Source, User Interface Editor, Function Tree Editor, and function panel windows.
- **Debugging Region**—Contains the Variables and Call Stack, Watch, Memory, and Resource Tracking windows. Use these windows to view and edit variable values, view the order of functions on the stack, program memory, and track the allocation and deallocation of resources during debugging.

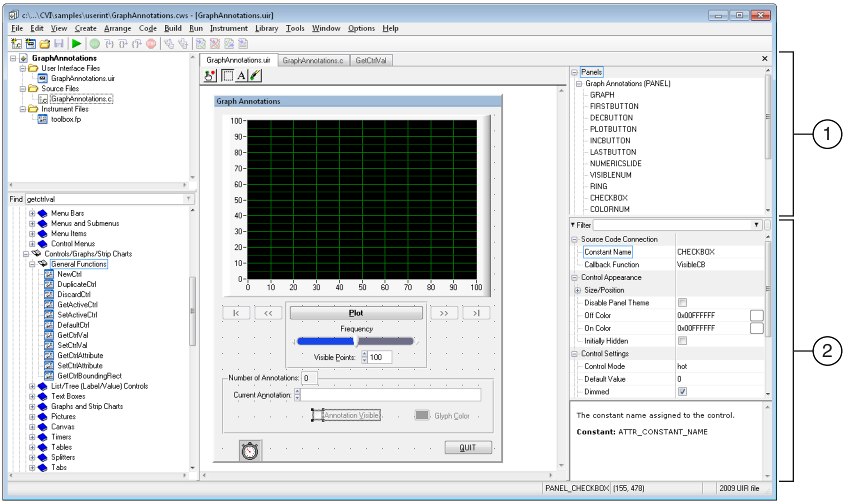
- **Output Region**—Contains the Build Output, Run-Time Errors, Source Code Control Errors, Debug Output, and Find Results windows. These windows contain lists of errors, output, and search matches.
- **Source Code Browser**—Contains browser overview information for selected files, functions, variables, data types, and macros in a program.
- **Status Bar**—Contains a status bar that displays the status of various aspects of the file, such as line and column number, save status, messages, and so on.

Select **Window»Release Window** to move a window that is contained within the Window Confinement Region, Debugging Region, Output Region, or Source Code Browser outside of the Workspace window.

Note You cannot release the Project Tree or Library Tree from the Workspace window. However, you can select **Options»Environment** and then enable the **Auto hide Project Tree and Library Tree** option to remove the Project Tree and Library Tree from the Workspace window when they are not in focus.

When you open a `.uir` file in the Workspace window, the User Interface Browser and Attribute Browser appear to the right of the User Interface Editor, which is shown in the following figure:

Figure 1-3. User Interface Browser and Attribute Browser



Source Code Connection	
Constant Name	CHECKBOX
Callback Function	VisibleCB
Control Appearance	
Size/Position	
Disable Panel Theme	☐
Off Color	0x00FFFFFF
On Color	0x00FFFFFF
Initially Hidden	☐
Control Settings	
Control Mode	hot
Default Value	0
Dimmed	✓

The constant name assigned to the control.

Constant: ATTR_CONSTANT_NAME

PANEL_CHECKBOX (155, 478) 2009 UIR file

1	User Interface Browser	2	Attribute Browser

Figure 1-3 displays the following areas in the Workspace window:

- **User Interface Browser**—Contains a tree view of the user interface objects, such as panels, controls, and menu bars, related to the selected `.uir`. Double-click a control, array of controls, or panel to highlight the object in the User Interface Editor and display the attributes related to the selected object in the Attribute Browser, where you can then edit the attributes.
- **Attribute Browser**—Use the Attribute Browser to edit attributes for the user interface objects. Enter an attribute name in the **Filter** text box at the top of the Attribute Browser to filter the attribute list down to those attributes with similar names. Alternately, you can click **Filter** to switch to the **Find** text box and search for a specific attribute within the browser.

Note The menus and toolbar buttons available within the LabWindows/CVI Workspace window differ depending on which window is active. To learn about what each menu item does, right-click the menu and select **Menu Help**. LabWindows/CVI launches the *LabWindows/CVI Help* topic that describes the items in the selected menu.

Standard Libraries

LabWindows/CVI provides a large set of built-in run-time libraries you can use to develop applications. You can browse the Library Tree or press <Ctrl-Shift-P> in a Source window to find a specific library function.

LabWindows/CVI includes the following standard libraries:

Table 1-1. Standard Libraries

Library	Description
User Interface Library	Functions for creating and controlling a graphical user interface
Analysis Library (Base Package)/Advanced Analysis Library (Full Development System)	Functions that operate on arrays to simulate and analyze large sets of numerical data quickly and efficiently
Formatting and I/O Library	Functions for inputting and outputting data to files and manipulating the format of data in a program
Utility Library	Functions that perform various operations, including using the system timer, managing disk files, launching another executable, and using multiple threads in a program

Table 1-1. Standard Libraries (Continued)

Library	Description
ANSI C Library	The ANSI C standard library functions
VXI Library	Functions for communicating with and controlling VXI devices
GPIB/GPIB 488.2 Library	Functions for communicating with and controlling devices on the GPIB
RS-232 Library	Functions for controlling multiple RS-232 ports using interrupt-driven I/O
VISA Library	Functions for controlling VXI, GPIB, serial, and other types of instruments
TCP Support Library	Functions that provide a platform-independent interface to the reliable, connection-oriented, byte-stream, network communication protocol
UDP Support Library	Functions that provide a platform-independent interface to the unicast, broadcast, and multicast capabilities of the User Datagram Protocol (UDP)
Internet Library (Full Development System)	Functions that communicate with and receive files and commands from remote servers
Network Variable Library	Functions for streaming data continuously between two LabWindows/CVI applications
Network Streams Library	Functions for streaming data continuously between two LabWindows/CVI applications
DDE Support Library	Functions that you can use to create an interface with other Windows applications using the Dynamic Data Exchange (DDE) standard
ActiveX Library	Functions that create and control ActiveX servers
DIAdem Connectivity Library	Functions that you can use to log test data in National Instruments DIAdem file format (`.tdm`)

Table 1-1. Standard Libraries (Continued)

Library	Description
TDM Streaming Library	Functions that store and retrieve test and measurement data using the `.tdms` file format. This file format is optimized for high performance data streaming
.NET Library	Functions that facilitate calling .NET assemblies
OpenMP Runtime Library (Full Development System)	Functions that you can use to create multithreaded applications using the OpenMP (Open Multi-Processing) model
Real-Time Utility Library	Functions for replicating a real-time (RT) system, configuring timing, creating and configuring trace sessions, and configuring RT targets

Note You must install the LabWindows/CVI Real-Time Module to gain access to the Real-Time Utility Library.

User Interface Development

Use LabWindows/CVI to develop GUIs that consist of panels, command buttons, pull-down menus, graphs, and many other controls and indicators. You can build a GUI interactively in the User Interface Editor or programmatically using the functions in the User Interface Library.

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

Generating a Program Shell with CodeBuilder

After you design a GUI in the User Interface Editor, you can use CodeBuilder to automatically generate a program shell based on the components in the GUI. CodeBuilder writes code for all control callback functions and creates a program skeleton that loads and displays GUI windows at program startup. To produce skeleton code after you design a GUI, select **Code»Generate**.

CodeBuilder saves development time by automating many of the common coding tasks required for writing a program. You use CodeBuilder later in this tutorial.

Developing and Editing Source Code

Use the Source window in LabWindows/CVI to develop C source files for projects. LabWindows/CVI 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 provides code generation tools that streamline source code development.

You can use the menu items in the Source window to edit files, debug code, compile files, and so on. You use Source window features in activities later in this tutorial.

Note For more information about the Source window, refer to **LabWindows/CVI Fundamentals»Writing Source Code** section in the *LabWindows/CVI Help*.

Instrument Control and Data Acquisition

You can use LabWindows/CVI to develop instrument control and data acquisition applications. LabWindows/CVI libraries provide functions for controlling GPIB, RS-232, serial, Ethernet, and National Instruments DAQ devices and modular instruments. LabWindows/CVI also provides interactive assistants you can use to generate code to communicate with different devices and to create and edit NI-DAQmx tasks.

Using the Instrument Control and Data Acquisition Libraries

LabWindows/CVI installs the GPIB/GPIB 488.2, VISA, and VXI libraries , but does not install the GPIB, NI-VISA, or NI-VXI drivers. While the GPIB/GPIB 488.2, VISA, and VXI libraries are listed in the Library Tree, you must install the drivers from the NI Device Drivers media to use the functions in an application.

Other driver-related libraries are not available in the Library Tree until you install the drivers. You can install drivers from `ni.com` or from the NI Device Drivers media.

For a list of hardware library documentation resources, refer to the *Related Documentation* section of the *About This Manual* chapter.

Using the DAQ Assistant

Use the NI DAQ Assistant to configure measurement tasks, channels, and scales. You also can use the DAQ Assistant to generate NI-DAQmx code from a task. To launch the DAQ Assistant from within the LabWindows/CVI environment, select **Tools»Create/Edit DAQmx Tasks**.

Note You must install NI-DAQmx from the NI Device Drivers media to use the DAQ Assistant.

Note For more information about using the DAQ Assistant, refer to the **Hardware Information»Data Acquisition»Where to Find Information about NI-DAQmx** section of the *LabWindows/CVI Help*.

Developing Instrument Drivers

If you plan to develop your own instrument driver, refer to the *IVI Driver Development Help*. This help file provides information about developing and adding instrument drivers to LabWindows/CVI. It is intended for programmers who develop instrument drivers to control programmable instruments such as GPIB, PXI, and RS-232 instruments. Also refer to the **LabWindows/CVI Fundamentals»Instrument Drivers** section of the *LabWindows/CVI Help* for fundamental instrument driver information you must consider if you create or modify a driver.

Learning about LabWindows/CVI

Complete the exercises in the remaining chapters of this tutorial to learn how to build, debug, and deploy applications in LabWindows/CVI. The following solution folder includes completed tutorial exercises you can use for reference:

Note The sample and exercise solutions in this folder are 64-bit compatible and might differ from code you write during these exercises.

`\Users\Public\Documents\National Instruments\CVI`*`version`*`\`
`tutorial\solution`

Note In the previous paths, *`version`* represents the version of LabWindows/CVI you are using.

The development process for the tutorial application includes the following steps:

1. Create a user interface in the User Interface Editor (Chapter 2, *Building a Graphical User Interface*).
2. Generate skeleton code for control callbacks using CodeBuilder (Chapter 2, *Building a Graphical User Interface*).
3. Add source code to generate and display a waveform (Chapter 3, *Using Function Panels and Libraries*).
4. Edit and debug the application (Chapter 4, *Editing and Debugging Tools*).
5. Develop a callback function to compute the maximum and minimum values of the waveform (Chapter 5, *Adding Analysis to Your Program*).
6. Create a distribution to deploy your application on another computer (Chapter 6, *Distributing Your Application*).

To view the example programs, navigate to the `\Users\Public\Documents\National Instruments\CVI`*`version`*`\samples` folder.

Refer to the following websites for additional support and information:

- `ni.com/cvi`—For general product information about LabWindows/CVI.
- `ni.com/examples`—For LabWindows/CVI example code and tutorials.
- `ni.com/forums`—To participate in discussion forums and exchange code with other LabWindows/CVI users around the world.
- `ni.com/cvinews`—To subscribe to the LabWindows/CVI newsletter or review the newsletter archive.
- `ni.com/cvi/community`—To participate in discussion forums and learn tips and tricks for working efficiently in LabWindows/CVI.

Where to Go Next

Complete the exercises in Chapter 2, *Building a Graphical User Interface*. Refer to the LabWindows/CVI documentation set for more information about the concepts presented in this manual. Use the *Guide to LabWindows/CVI Documentation* topic in the *LabWindows/CVI Help* to learn more about and access the documents in the LabWindows/CVI documentation set. To launch the *LabWindows/CVI Help*, select **Help»Contents**.

2

Building a Graphical User Interface

In the remaining chapters of this tutorial, you develop a project that consists of a GUI controlled by a C source file, which acquires and displays a waveform on the GUI. In this chapter, you learn to design a user interface with the User Interface Editor.

Project Templates

Using project and file templates can help reduce the time and effort required to configure a new project or file. The template includes the basic settings for the new project or file and any preliminary text to include by default, such as standard comments or headings. For more information about project templates, refer to the **Creating Applications»Managing Projects» Creating Projects and Files from Templates»New Project and File Templates** section in the *LabWindows/CVI Help*.

Selecting a Project Template

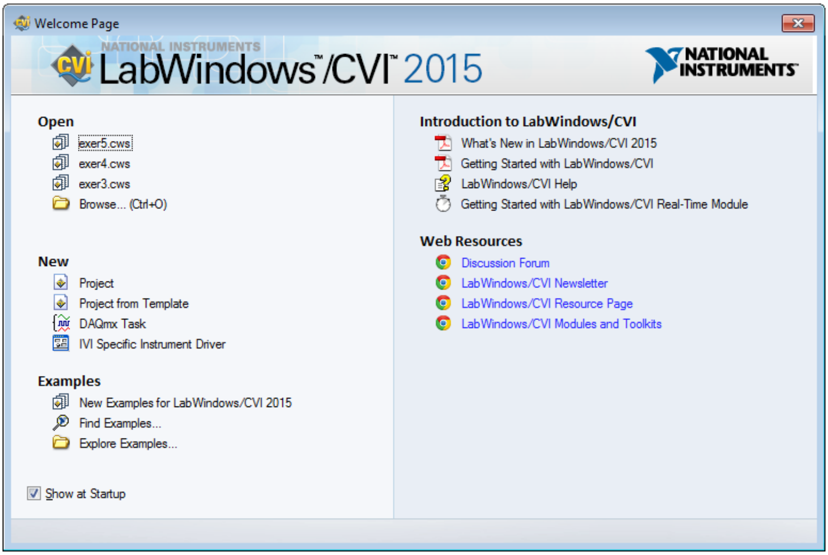
Complete the following steps to select and set up a project template for the sample project:

1. Launch LabWindows/CVI by selecting **Start»All Programs»National Instruments» LabWindows CVI *version*»LabWindows CVI *version***.

   Note **(Windows 8)** Click NI Launcher and select **LabWindows/CVI *version*** in the window that appears.

2. The Welcome Page, shown in the following image, is displayed when you open LabWindows/CVI for the first time. It displays options for opening files and examples, helpful resources, and recently opened files. Select **Project from Template** to open the New Project from Template dialog box.

Figure 2-1. LabWindows/CVI Welcome Page


Welcome Page
NATIONAL INSTRUMENTS
LabWindows™/CVI™ 2015
NATIONAL INSTRUMENTS
Open
exer5.cws
exer4.cws
exer3.cws
Browse... (Ctrl+O)
New
Project
Project from Template
DAQmx Task
IVI Specific Instrument Driver
Examples
New Examples for LabWindows/CVI 2015
Find Examples...
Explore Examples...
Show at Startup
Introduction to LabWindows/CVI
What's New in LabWindows/CVI 2015
Getting Started with LabWindows/CVI
LabWindows/CVI Help
Getting Started with LabWindows/CVI Real-Time Module
Web Resources
Discussion Forum
LabWindows/CVI Newsletter
LabWindows/CVI Resource Page
LabWindows/CVI Modules and Toolkits

Note If you disable the Welcome Page, you see an empty workspace when you start LabWindows/CVI. Select **File»New»Project from Template** to open the New Project from Template dialog box.

3. Select **User Interface Application**.
4. Change **Project name** to `sample1`.
5. Change the **Project folder** to the tutorial folder.

   `\Users\Public\Documents\National Instruments\CVI`*`version`*`\tutorial\`.
6. Verify that **Add this project to the current workspace** is not selected.
7. Click **OK**.

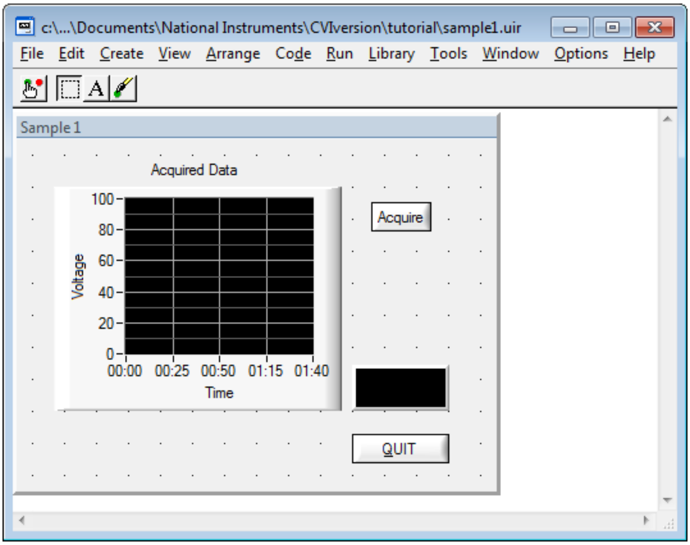
User Interface Editor

In the User Interface Editor, you can select a number of different controls from the **Create** menu and position them on the panels you create. You can customize each control through a series of dialog boxes in which you set attributes for the control appearance, source code connections, and label appearance.

Building a User Interface Resource (.uir) File

Complete the following steps to design the user interface for the sample project, as shown in Figure 2-2.

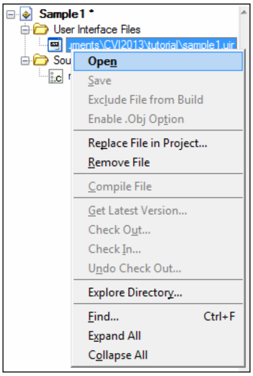
Figure 2-2. sample1.uir


c:\...\Documents\National Instruments\CVIversion\tutorial\sample1.uir
File Edit Create View Arrange Code Run Library Tools Window Options Help
Sample 1
Acquired Data
Voltage
100
80
60
40
20
0
00:00 00:25 00:50 01:15 01:40
Time
Acquire
QUIT

Editing a .uir File

1. In the Project Tree, right-click `sample1.uir`. Select **Open** to open the file, shown in the following figure.

Figure 2-3. Opening `sample1.uir`



2. The Attribute Browser is located to the right of the User Interface Editor in the Workspace window, below the User Interface Browser. In the Attribute Browser, the **Find** text box highlights attributes in the list and the **Filter** text box displays only matching attributes in the list. You can change between finding or filtering attributes in the list by clicking **Find** or **Filter**, shown in the following figures.

   Use the **Find** or the **Filter** text box to locate the **Constant Name** control in the Attribute Browser. Notice that **Constant Name** is set to `PANEL` and **Callback Function** is set to `panelCB`. These are the settings from the project template.

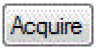
Note In the User Interface Editor, select the control(s) or panel you want to edit to display the associated attributes in the Attribute Browser.

3. Enter `Sample 1` for the **Title** and press the <Enter> key to commit any attribute value changes.

You also can edit control and panel attribute values in the Edit Control or Edit Panel dialog box, respectively. Double-click a control or panel in the `.uir` to open the edit dialog box. Notice that some attributes have slightly different names in the edit dialog box compared to the names in the Attribute Browser.

Adding Command Buttons

1. Select **Create»Command Button»Square Command Button**. LabWindows/CVI places a button labeled **OK** on the panel.
2. To edit the button attributes, select the button. The attributes related to the button appear in the Attribute Browser.
3. To change the label on the command button, enter `Acquire` in place of __`OK` in **Label Text**. The change is displayed in the User Interface Editor, shown in the following figure.

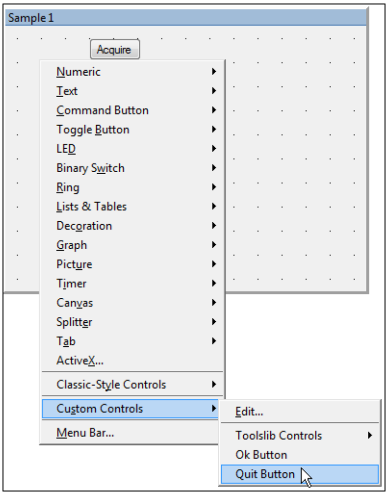


Note If you type a double underscore before any letter in **Label Text**, the letter is underlined on the user interface. The user can select the control by pressing <Alt> and the underlined letter, provided that no accessible menu bars contain a menu with the same underlined letter.

4. Assign a constant name to the button. In the C source code you use this constant name to identify the button. Change the default **Constant Name** to `ACQUIRE`.

5. Assign a function name that the program calls when a user clicks the **Acquire** button. Enter `AcquireData` as the **Callback Function**. In Chapter 3, *Using Function Panels and Libraries*, you write the source code for the `AcquireData` function.
6. (Optional) Customize the label font appearance by changing the values in the **Label Bold** field, the **Label Character Set** field, and so on.
7. To add the **QUIT** button, ensure the `.uir` file has focus, right-click the panel, and select **Custom Controls»Quit Button**, shown in the following figure. Custom controls are frequently used control configurations. The **QUIT** button already has a callback function, `QuitCallback`, assigned. It is not necessary to modify the default settings for the **QUIT** button.

Figure 2-4. Navigating to the Quit Button

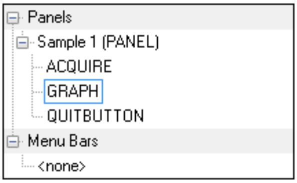


Adding a Graph Control

1. Right-click the Sample 1 panel and select **Graph»Graph**. LabWindows/CVI places a graph control labeled Untitled Control on the panel.
2. To size the panel, click and drag one of its corners. Use the commands in the **Edit** menu and the **Arrange** menu to cut, copy, paste, align, and space user interface controls in the editor. You also can use the grid lines on the panel to align the controls.

3. To locate control attributes using the User Interface Browser located above the Attribute Browser, click **GRAPH** in the User Interface Browser. Notice that the graph is highlighted in the User Interface Editor, shown in the following figure. The attributes available in the Attribute Browser are now attributes associated with the graph control.

Figure 2-5. Attribute Browser



To customize the graph attributes, enter the following values in the Attribute Browser:

a. Use the **Find** text box to locate the **Constant Name** attribute. If you prefer, use the **Filter** text box accessible by toggling the **Find** label.

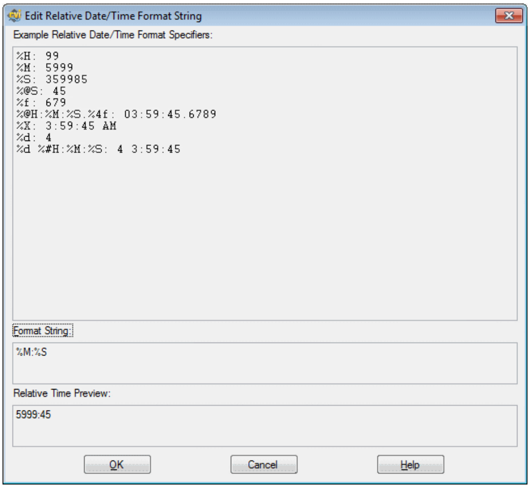
   Enter `WAVEFORM` as the **Constant Name**.

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

b. Enter `Acquired Data` as the **Label Text**.
c. Enter `Time` for **X Name**.
d. To display time relative to the start of the application, set **X Format** to **relative time**.

e. Click the **...** button in the value column of **X Axis Date Time Format String** to open the Edit Relative Date/Time Format String dialog box. To display time in minutes and seconds, delete `%H:` from the **Format String** field, shown in the following figure, and click **OK**.

Figure 2-6. Edit Relative Date/Time Format String Dialog Box


Edit Relative Date/Time Format String

Example Relative Date/Time Format Specifiers:

```
%H: 99
%M: 5999
%S: 359985
%@S: 45
%f: 679
%@H:%M:%S.%4f: 03:59:45.6789
%X: 3:59:45 AM
%d: 4
%d %#H:%M:%S: 4 3:59:45
```

Format String:

```
%M:%S
```

Relative Time Preview:

```
5999:45
```

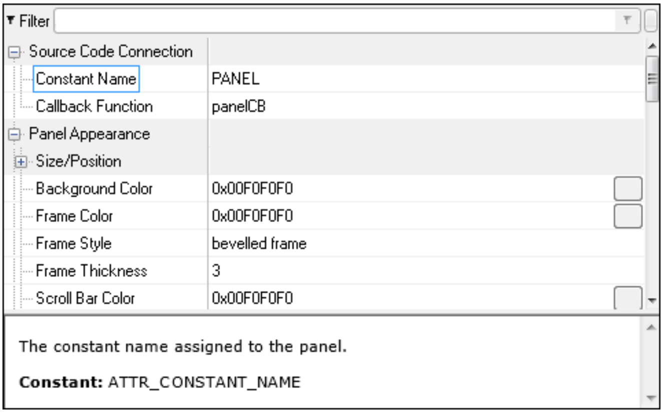
f. Set **Y Name** to `Voltage`.

4. Verify that the completed user interface looks like the one shown in Figure 2-2.

Source Code Connection

After you design a user interface in the User Interface Editor, you can write C source code to control the GUI. To connect elements on the user interface to the source code, you must assign a constant name to each panel, menu, and control on your user interface, shown in Figure 2-7. Then, you can use those names in the C source code to differentiate the controls on the GUI. You also can assign a callback function to a control that is called automatically when you operate that control during program execution. Use the Attribute Browser to associate a constant name and a callback function with that control.

Figure 2-7. Assigning a Constant Name



Source Code Connection	
Constant Name	PANEL
Callback Function	panelCB
Panel Appearance	
Size/Position	
Background Color	0x00F0F0F0
Frame Color	0x00F0F0F0
Frame Style	bevelled frame
Frame Thickness	3
Scroll Bar Color	0x00F0F0F0

The constant name assigned to the panel.

Constant: ATTR_CONSTANT_NAME

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

CodeBuilder

After you complete the `.uir` file, you can use CodeBuilder to expand on code in the project template source file by generating the skeleton code for the remaining callback functions for the controls on your panel.

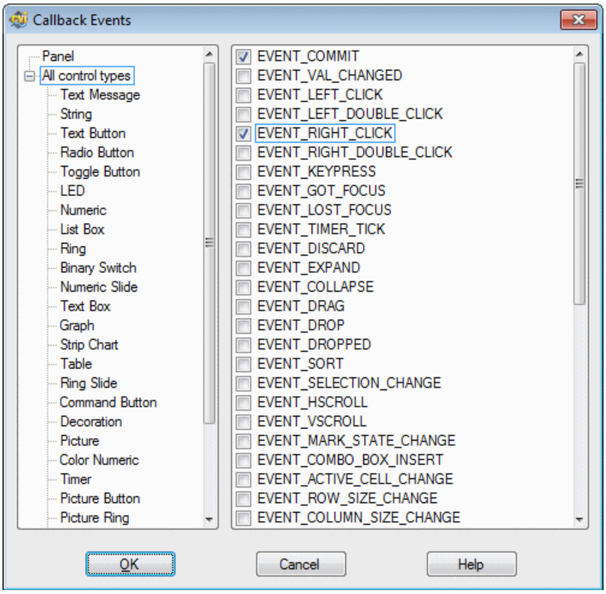
Note For more information about CodeBuilder, refer to the **Creating Applications»Developing a Graphical User Interface»Generating Code from the GUI** section of the *LabWindows/CVI Help*.

Completing the Program Shell with CodeBuilder

Now that you have built a GUI in the User Interface Editor, use CodeBuilder to complete the remainder of the program shell for your GUI. Generate the skeleton code for the control callbacks.

1. To select default control events for your application, select **Code»Preferences»Default Events** to open the Callback Events dialog box. Select **All control types** from the left tree and select `EVENT_COMMIT` and `EVENT_RIGHT_CLICK`, shown in the following figure. Verify that no other events are selected. Click **OK**.

Figure 2-8. Callback Events Dialog Box



Callback Events

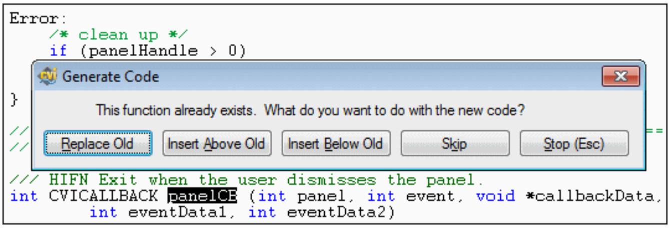
- Panel
- All control types
  - Text Message
  - String
  - Text Button
  - Radio Button
  - Toggle Button
  - LED
  - Numeric
  - List Box
  - Ring
  - Binary Switch
  - Numeric Slide
  - Text Box
  - Graph
  - Strip Chart
  - Table
  - Ring Slide
  - Command Button
  - Decoration
  - Picture
  - Color Numeric
  - Timer
  - Picture Button
  - Picture Ring

- [✓] EVENT_COMMIT
- [ ] EVENT_VAL_CHANGED
- [ ] EVENT_LEFT_CLICK
- [ ] EVENT_LEFT_DOUBLE_CLICK
- [✓] EVENT_RIGHT_CLICK
- [ ] EVENT_RIGHT_DOUBLE_CLICK
- [ ] EVENT_KEYPRESS
- [ ] EVENT_GOT_FOCUS
- [ ] EVENT_LOST_FOCUS
- [ ] EVENT_TIMER_TICK
- [ ] EVENT_DISCARD
- [ ] EVENT_EXPAND
- [ ] EVENT_COLLAPSE
- [ ] EVENT_DRAG
- [ ] EVENT_DROP
- [ ] EVENT_DROPPED
- [ ] EVENT_SORT
- [ ] EVENT_SELECTION_CHANGE
- [ ] EVENT_HSCROLL
- [ ] EVENT_VSCROLL
- [ ] EVENT_MARK_STATE_CHANGE
- [ ] EVENT_COMBO_BOX_INSERT
- [ ] EVENT_ACTIVE_CELL_CHANGE
- [ ] EVENT_ROW_SIZE_CHANGE
- [ ] EVENT_COLUMN_SIZE_CHANGE

In the main tutorial, you work only with the `EVENT_COMMIT` event. In Chapter 7, *Additional Exercises*, you develop code to display help when a user right-clicks a GUI control. For a complete list of events, refer to the **Library Reference»User Interface Library»Events** topic of the *LabWindows/CVI Help*.

2. Select **Code»Generate»All Callbacks**. By default, CodeBuilder generates the `EVENT_RIGHT_CLICK` and `EVENT_COMMIT` events for every panel or control for which you define a callback function.
3. The Generate Code dialog box appears with the `panelCB` callback function highlighted in the Source window, shown in the following figure. The project template provides the `panelCB` callback function, so you do not need CodeBuilder to generate it. Click **Skip**. CodeBuilder proceeds and generates the callback functions for all of the controls.

Figure 2-9. Generate Code Dialog Box



```
Error:
    /* clean up */
    if (panelHandle > 0)
}
//
//
/// HIFN Exit when the user dismisses the panel.
int CVICALLBACK panelCB (int panel, int event, void *callbackData,
        int eventData1, int eventData2)
```

Generate Code

This function already exists. What do you want to do with the new code?

4. Insert a line after `case EVENT_COMMIT:` in the `QuitCallback` function with the following code:

```
QuitUserInterface(0);
```

Note The project template provides only one option for closing the panel, clicking the X button in the upper right corner. By inserting a call to `QuitUserInterface` in `QuitCallback`, you add a second option for closing the panel, clicking the **QUIT** button.

Analyzing the Source Code

The source code that you generated for the Sample 1 program is skeleton code. You must add code to this skeleton that determines how the program responds when it generates events. The program you generated consists of three functions.

main Function

Completing the `main` function is the first step you must take when you build your own applications. The `main` function is shown in the following code:

```
int main (int argc, char *argv[])
{
    int error = 0;

    /* initialize and load resources */
    nullChk (InitCVIRTE (0, argv, 0));
    errChk (panelHandle = LoadPanel (0, "sample1.uir", PANEL));

    /* display the panel and run the user interface */
    errChk (DisplayPanel (panelHandle));
    errChk (RunUserInterface ());

Error:
    /* clean up */
    if (panelHandle > 0)
        DiscardPanel (panelHandle);
    return 0;
}
```

Note The LabWindows/CVI macros `errChk` and `nullChk` provide convenient error-handling. For more information about these macros, refer to `toolbox.h`.

To allow users to operate the user interface that you created, your program must perform the following steps:

- `LoadPanel` loads the panel from the `.uir` file into memory.
- `DisplayPanel` displays the panel on the screen.
- `RunUserInterface` allows LabWindows/CVI to begin sending events from the user interface to the C program you are developing. This function does not return until the program calls `QuitUserInterface`.

When you no longer need the user interface, call `DiscardPanel` to remove the panel from memory and from the screen.

AcquireData Function

The `AcquireData` function automatically executes whenever you click **Acquire** on the user interface. You add to this function later in this tutorial 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;
}
```

Note Notice that the callback function returns 0. User callbacks must always return 0 unless they intend to swallow the event to which they are responding. To swallow the event, the callback should return 1. This tutorial does not use callbacks that swallow events. Refer to the **Library Reference»User Interface Library»Events» Swallowing Events** topic in the *LabWindows/CVI Help* for more information about swallowing events, including a list of events that are swallowable.

QuitCallback Function

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

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

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}
```

Running the Generated Code

The code you generated using CodeBuilder is syntactically and programmatically correct code that compiles and runs before you add to it. Select **Run»Debug sample1.exe** to run the generated code. The program displays the user interface panel that is shown in Figure 2-2 and exits when you press the **QUIT** button.

Where to Go Next

After you debug `sample1.exe`, ensure that the application matches Figure 2-2. Next, proceed to Chapter 3 in which you will use LabWindows/CVI function panels to generate code.

3

Using Function Panels and Libraries

In this chapter of the tutorial, you use LabWindows/CVI function panels to generate code. You complete the source code that plots an array with a sine pattern on the graph control you created in Chapter 2, *Building a Graphical User Interface*. If you have not completed the exercises in Chapter 2, go back and do so now.

Function Panel Fundamentals

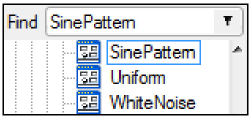
A function panel is a graphical view of a library function in LabWindows/CVI. Function panels serve several important purposes.

- With function panels, you can execute each LabWindows/CVI function interactively before incorporating it into the program. With this feature, you can experiment with the parameter values until you are satisfied with the operation of the function.
- Function panels provide help that explains the purpose of each function in the LabWindows/CVI libraries and of each parameter in the function call.
- Function panels generate code automatically so that you can insert the function call syntax into your program source code.

Accessing Function Panels

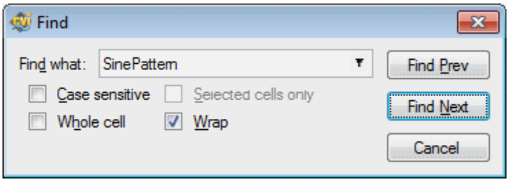
The Library Tree includes function panels for all of the libraries in LabWindows/CVI. You can scan quickly through the hierarchy of the library to find a given function. Alternatively, you can use the **Find** text box located above the Library Tree and enter the name of the function, shown in the following figure.

Figure 3-1. Find Text Box



For advanced searching options, right-click the Library Tree, select **Find**, enter the name of the function, and select from several search options such as case-sensitivity, wrapping text, and so on, shown in the following figure.

Figure 3-2. Find Dialog Box



Function Panel Controls

The controls on the function panel represent parameters. Enter values in the controls to specify parameter values. Some controls have **...** buttons next to them that provide additional dialog boxes to select input for parameters.

Function Panel Help

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

Table 3-1. Displaying Function Panel Help

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

Note A function must be documented in the *LabWindows/CVI Help* to have combined help available through its function panel.

Generating an Array of Data

When a user clicks **Acquire**, the program generates a random number using the ANSI C `srand` and `rand` functions and then uses that number as the amplitude for the sine pattern. Complete the following steps to implement the simulated data generation in the `AcquireData` callback function:

1. Open `sample1.c`, if it is not already open.
2. In the `AcquireData` function, on the line following `case EVENT_COMMIT`:, enter the following lines of code to generate the random numbers.

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

Note Refer to `sample1.c` in the solution folder for an example of the previous code that is 64-bit compliant.

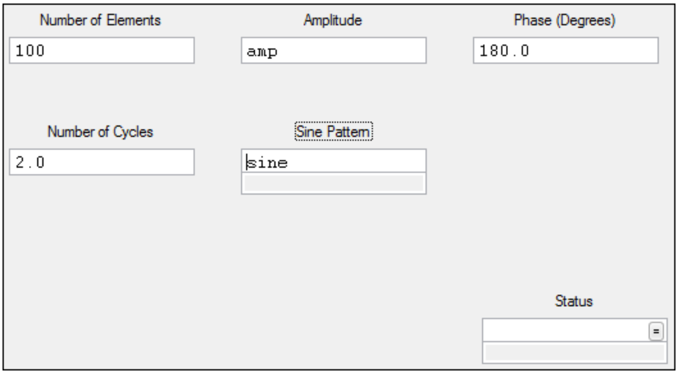
3. Position the cursor on a blank line immediately following `amp = rand ()/32767.0`.
4. Enter `SinePattern` in the **Find** text box above the Library Tree to locate the Sine Pattern function panel and press the <Enter> key to highlight the function in the Library Tree. Then press the <Enter> key again to open the function panel.

Note If LabWindows/CVI cannot find a match, right-click the Library Tree and select **Show Function Names**. Then repeat step 4.

5. Select **Code»Set Target File**. Select **sample1.c** and click **OK**.
6. Enter `100` in the **Number of Elements** control.
7. Enter `amp` in the **Amplitude** control. Select **Code»Declare Variable** and ensure that only the **Add declaration to current block in target file “sample1.c”** option is enabled. Click **OK**.
8. Enter `180.0` in the **Phase (Degrees)** control.
9. Enter `2.0` in the **Number of Cycles** control.
10. Enter `sine` in the **Sine Pattern** control. Select **Code»Declare Variable**.

11. In the Declare Variable dialog box, enter `100` as the **Number of Elements** and ensure that **Add declaration to top of target file “sample1.c”** option. Click **OK**. Ensure the function panel matches the following image.

Figure 3-3. Function Panel



Number of Elements: 100
Amplitude: amp
Phase (Degrees): 180.0
Number of Cycles: 2.0
Sine Pattern: sine

12. Select **Code»Insert Function Call**. LabWindows/CVI pastes the `SinePattern` function from the function panel into the `sample1.c` source code at the position of the text cursor.

Tip In the Source window, you can place your cursor anywhere in a LabWindows/CVI library function call and then select **View»Recall Function Panel** to open the function panel for the selected function. When you recall a function panel, the controls automatically reflect the state of the function call in the Source window.

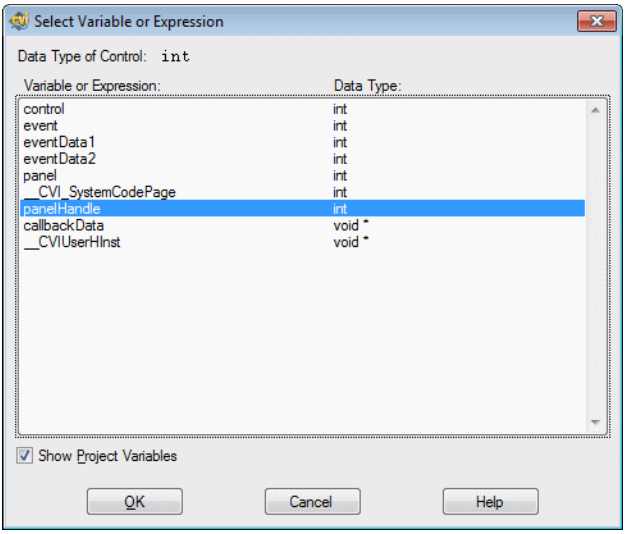
Building the PlotY Function Call Syntax

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

1. Position the cursor in the Source window on a blank line immediately following the `SinePattern` function call within the `AcquireData` function.
2. Type `PlotY` and then press <Ctrl-P> to open the Plot Y function panel.

3. In the **Panel Handle** control, select **Code»Select Variable**. Enable the **Show Project Variables** option. The dialog box contains a list of variable names used in your program. Choose **panelHandle** from the list, shown in the following figure, and click **OK**.

Figure 3-4. Function Panel



Select Variable or Expression

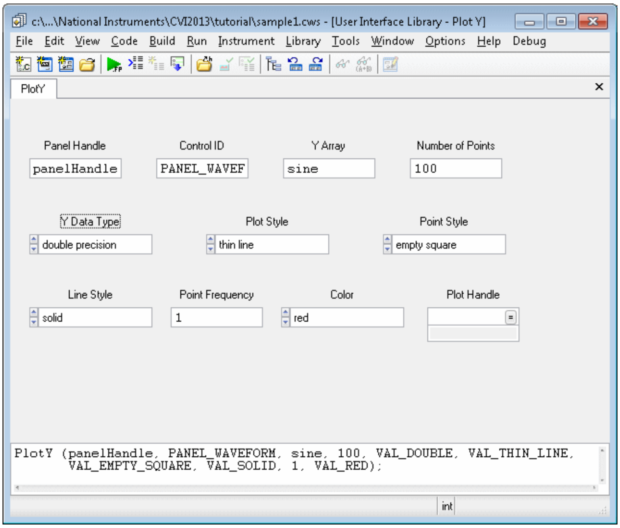
Data Type of Control: `int`

Variable or Expression:	Data Type:
control	int
event	int
eventData1	int
eventData2	int
panel	int
__CVI_SystemCodePage	int
panelHandle	int
callbackData	void *
__CVIUserHInst	void *

☑ Show Project Variables

4. For the **Control ID** control, you must specify the constant name assigned to the graph control. While the cursor is in **Control ID**, press <Enter> to open a dialog box with a list of the constant names for controls in the `.uir` files in the workspace. Verify **sample1.uir** is selected in the **User Interface Resource files** section, select **PANEL_WAVEFORM** from the list of constants, and click **OK**.
5. Type `sine` in the **Y Array** control. This name indicates which array in memory the program displays on the graph.
6. Type `100` in the **Number of Points** control. This number indicates the number of elements in the array to plot.
7. For **Y Data Type**, click the control to display a drop-down menu of possible data types. Select **double precision**. When the Plot Y function panel matches the one in Figure 3-5, proceed to the next step.

Figure 3-5. Completed Plot Y Function Panel


c:\...\National Instruments\CVI2013\tutorial\sample1.cws - [User Interface Library - Plot Y]
File Edit View Code Build Run Instrument Library Tools Window Options Help Debug
PlotY
int

Panel Handle: panelHandle
Control ID: PANEL_WAVEF
Y Array: sine
Number of Points: 100
Y Data Type: double precision
Plot Style: thin line
Point Style: empty square
Line Style: solid
Point Frequency: 1
Color: red
Plot Handle:

```
PlotY (panelHandle, PANEL_WAVEFORM, sine, 100, VAL_DOUBLE, VAL_THIN_LINE,
       VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
```

8. Select **Code»Insert Function Call** to paste the `PlotY` function call into the source code. LabWindows/CVI displays a message that states text is selected on the current line. Click **Replace** to replace the `PlotY` you typed with the complete function call.
9. Confirm that the `AcquireData` function matches the following source code:

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

            break;
    }
    return 0;
}
```

10. Save the source file.

Running the Completed Project

You now have a completed project, saved as `sample1.prj`. Select **Run»Debug sample1.exe** to execute the code. If prompted, click **Yes** to add `ansi_c.h` and `analysis.h` to the top of the file.

Note If LabWindows/CVI prompts you to add a reference to a header file, LabWindows/CVI also displays an error indicating that you are calling a function that is not defined. When you add the header file, LabWindows/CVI can find the function definition and no error occurs when you next execute the code.

When you run your program, the following actions take place:

1. LabWindows/CVI compiles the source code from `sample1.c` and links with the appropriate libraries in LabWindows/CVI.
2. When the program starts, LabWindows/CVI launches the user interface, ready for keyboard or mouse input.
3. When you click **Acquire**, LabWindows/CVI passes the event to the `AcquireData` callback function.
4. The `AcquireData` function generates an array of data and plots it on the graph control on the user interface.
5. When you click **QUIT**, LabWindows/CVI passes the event to the `QuitCallback` function, which halts the program.

Where to Go Next

Complete the tutorial in Chapter 4 in which you will edit and debug in the LabWindows/CVI environment.

4 Editing and Debugging Tools

This chapter uses the project you developed in Chapter 3, *Using Function Panels and Libraries*. If you did not proceed directly from Chapter 3, go back and do so now.

In this chapter, you become acquainted with LabWindows/CVI editing and debugging tools.

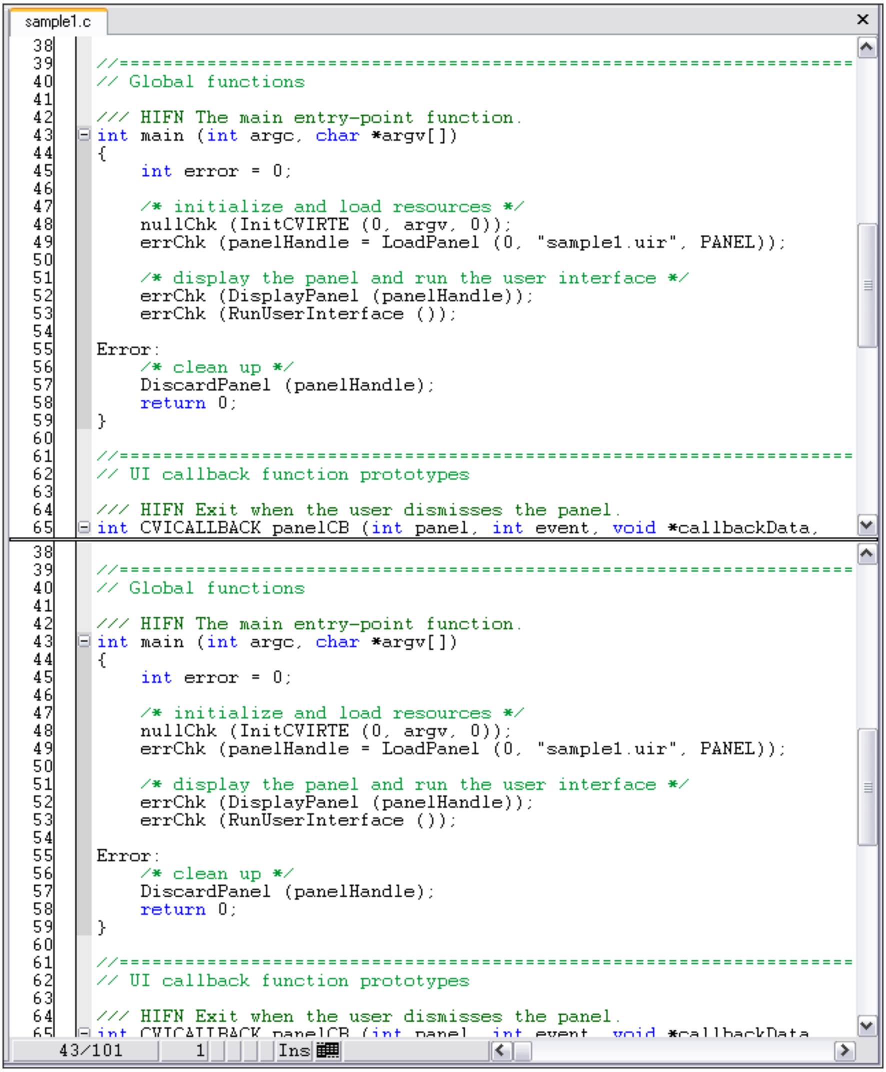
Editing Tools

The LabWindows/CVI Source window has a number of quick editing features that are helpful when you work with source files. Complete the following steps to view some of the editing features available in LabWindows/CVI:

1. Open `sample1.c` if it is not already open. Select **View»Line Numbers** to display a column to the left of the window that shows line numbers.
2. The programs you develop in LabWindows/CVI often refer to other files, such as header files or user interface files. To view these additional files quickly, place the cursor on the filename in the source code and select **File»Open Quoted Text**, press <Ctrl-U>, or right-click the filename and select **Open Quoted Text**.

   Place the cursor on the `userint.h` filename in `sample1.c` and press <Ctrl-U>. LabWindows/CVI opens the `userint.h` header file in a separate Source window. Scroll through and then close the header file.
3. If you want to view a portion of your source code while you make changes to another area of the source code in the same file, you can split the window into top and bottom halves called *subwindows*, as shown in Figure 4-1.
4. To split the window, click and drag the double line at the top of the Source window to the middle of the screen. Notice how each half of the window scrolls independently to display different areas of the same file simultaneously.

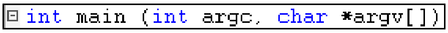
Figure 4-1. Split Source Window



```

//=====================================================================
// Global functions

/// HIFN The main entry-point function.
int main (int argc, char *argv[])
{
    int error = 0;

    /* initialize and load resources */
    nullChk (InitCVIRTE (0, argv, 0));
    errChk (panelHandle = LoadPanel (0, "sample1.uir", PANEL));

    /* display the panel and run the user interface */
    errChk (DisplayPanel (panelHandle));
    errChk (RunUserInterface ());

Error:
    /* clean up */
    DiscardPanel (panelHandle);
    return 0;
}

//=====================================================================
// UI callback function prototypes

/// HIFN Exit when the user dismisses the panel.
int CVICALLBACK panelCB (int panel, int event, void *callbackData,
```

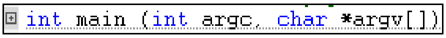
```

//=====================================================================
// Global functions

/// HIFN The main entry-point function.
int main (int argc, char *argv[])
{
    int error = 0;

    /* initialize and load resources */
    nullChk (InitCVIRTE (0, argv, 0));
    errChk (panelHandle = LoadPanel (0, "sample1.uir", PANEL));

    /* display the panel and run the user interface */
    errChk (DisplayPanel (panelHandle));
    errChk (RunUserInterface ());

Error:
    /* clean up */
    DiscardPanel (panelHandle);
    return 0;
}

//=====================================================================
// UI callback function prototypes

/// HIFN Exit when the user dismisses the panel.
int CVICALLBACK panelCB (int panel, int event, void *callbackData
```

5. Drag the dividing line between the two subwindows back to the top to make a single window again.
6. If you want a cleaner view of your code, you can collapse certain regions. Click the minus button to the left of the `main` function in `sample1.c`, shown in the following figure.



```
int main (int argc, char *argv[])
```

The function collapses into a single line of code with a dotted line beneath it, shown in the following figure. The minus button is now a plus button, signaling there is hidden collapsed code. Click the plus button to reveal the code.



```
⊞ int main (int argc, char *argv[])
```

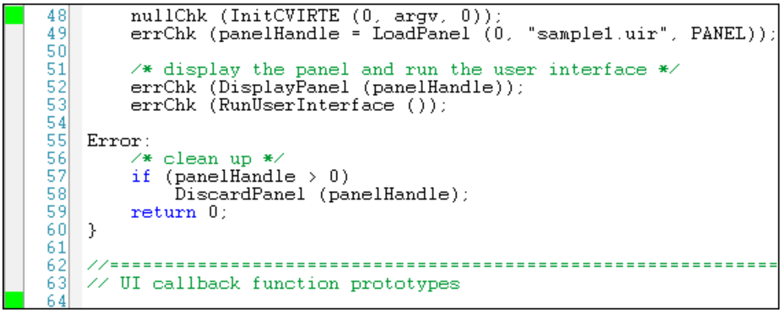
Tip To recursively collapse the region and any subregions of code, right-click the minus button and select **Collapse Region and Subregions**.

7. Place your mouse over the collapsible region to highlight the region in the column that corresponds to the code block. The highlight persists until you move your mouse to another part of the collapsible region. If you move your mouse away from the collapsible region and into the source code, the highlight persists in a lighter shade.

Note LabWindows/CVI defines collapsible regions for multiline code blocks delimited by curly braces or multiline comments. For more information about collapsible regions, refer to the **LabWindows/CVI Fundamentals»Writing Source Code»Source Window Overview»Collapsible Regions** section in the *LabWindows/CVI Help*.

8. To quickly move to a particular line of code in your source file, select **View»Line** to open the Line dialog box. Then enter the line number in the **Go to Line** control and click the **OK** button.
9. To set a tag, place the cursor on line 48. Select **View»Toggle Tag**. A green square appears in the left-hand column of the Source window. Place the cursor on line 64 of the Source window and add another tag, shown in the following figure.

Figure 4-2. . Adding Tags



```
48      nullChk (InitCVIRTE (0, argv, 0));
49      errChk (panelHandle = LoadPanel (0, "sample1.uir", PANEL));
50
51      /* display the panel and run the user interface */
52      errChk (DisplayPanel (panelHandle));
53      errChk (RunUserInterface ());
54
55  Error:
56      /* clean up */
57      if (panelHandle > 0)
58          DiscardPanel (panelHandle);
59      return 0;
60  }
61
62  //==============================================================
63  // UI callback function prototypes
64
```

10. Press <F2> to move between tags. Select **View»Clear Tags**, make sure all of the tags are checked, and then click **OK** to remove the tags from the source file.
11. To find specific text in the code, select **Edit»Find** to enter the text you want to locate and specify various searching preferences. Enter `panelHandle` in the **Find what** control and leave the remaining controls set to their default values. Then click **Find Next**.

LabWindows/CVI highlights the first match in the text and displays a list of all matches in the Find Results window. Click an entry in the Find Results window to locate the corresponding text in the Source window.

12. To complete a quick search, select **Edit»Quick Search** and type `argc`. Notice that LabWindows/CVI finds matches of the letters you type. The selection changes as you type more letters.

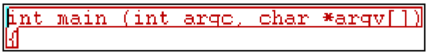
Step Mode Execution

In LabWindows/CVI, you can compile and build a program in debug configuration or release configuration by selecting the configuration from the **Build»Configuration** submenu. The debug configuration executes similarly to the release version of the program but also offers you tools for debugging the program, including breakpoints and step mode execution. After debugging your program, select a release configuration to remove debugging information and increase the execution speed of your program. The following debugging exercises use debug configuration.

Note Refer to the **Creating Applications»Managing Projects»Setting the Project Configuration** topic in the *LabWindows/CVI Help* for more information about project configurations.

Step mode execution is a useful run-time tool for debugging programs. To step through `sample1.c`, complete the following steps:

1. Select **Run»Break on»First Statement** to stop execution at the first statement in the source code.
2. Select **Run»Debug sample1.exe** to begin program execution. After the program compiles, the `main` function line in the program is highlighted in the Source window, shown in the following figure. This indicates that program execution is currently suspended.



```
int main (int argc, char *argv[])
{
```

3. To execute the highlighted line, select **Run»Step Into**.

Tip Alternatively, use the icons in the toolbar and the shortcut key combinations listed in Table 4-1 to execute these commands.

Table 4-1. Quick Keys for Step Mode Execution

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

4. To find the definition of the `SinePattern` function, place the cursor on the function in `sample1.c` and select **Edit»Go to»Go to Definition**.

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

   In this case, LabWindows/CVI opens `analysis.h` and highlights the `SinePattern` function declaration. To return to your previous source code location, select **Edit»Go to» Go Back**.

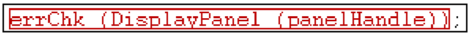
Tip Many of the commands in this exercise also are available in the Source window context menu. Right-click within the Source window to view the available commands.

5. Use the **Step Into** button to begin stepping through the program. Notice that when the `main` function is executed, the highlighting moves to the function and traces the instructions

inside the function. Continue to step through the program until the following statement is highlighted:

```
errChk(DisplayPanel (panelHandle));
```

6. Place the cursor on the line with the call to `DiscardPanel (panelHandle);`. Select **Run»Set Next Statement** to select the next statement to execute. The highlighting moves to that line.



```
errChk (DisplayPanel (panelHandle));
```

7. Press <F5> to continue program execution. Notice that the program exits without having run the user interface because the program execution skipped over the `RunUserInterface` function call.

Breakpoints

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

- **Fixed Breakpoint**—Insert a breakpoint at a particular location in the Source window. You can turn breakpoints on or off even while your program is executing.
- **Instant Breakpoint**—When an application is running, press <Ctrl-F12> while a window is active in the LabWindows/CVI environment.
- **Breakpoint on Library Errors**—Select **Run»Break on»Library Errors** to cause LabWindows/CVI to pause at a particular location when a library function returns an error.
- **Conditional Breakpoint**—Cause LabWindows/CVI to pause at a particular location when a user-specified condition becomes true.
- **Programmatic Breakpoint**—In your code, call the `Breakpoint` function.
- **Watch Expression Breakpoint**—Cause LabWindows/CVI to pause when the value of a watch expression changes.

Fixed Breakpoints

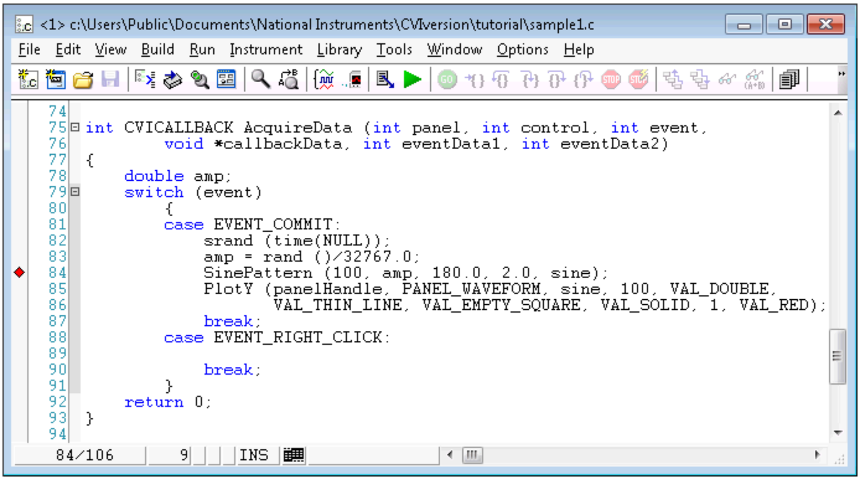
To insert a breakpoint at a specific location in your source code, click in the left column of the Source window on the line on which you want to suspend execution. Complete the following steps to insert a breakpoint inside the `AcquireData` function:

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

```
SinePattern (100, amp, 180.0, 2.0, sine);
```

A red diamond, which represents a breakpoint, appears beside that line as shown in Figure 4-3.

Figure 4-3. Breakpoint Beside a Line of Code


<1> c:\Users\Public\Documents\National Instruments\CVIversion\tutorial\sample1.c
File Edit View Build Run Instrument Library Tools Window Options Help
84/106
9
INS

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

            break;
        }
    return 0;
}
```

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

4. Begin execution of the program by selecting **Run»Debug sample1.exe**. Click **Acquire** to generate a commit event for `AcquireData`. When LabWindows/CVI encounters the breakpoint during execution, it suspends program execution and highlights the line where you inserted the breakpoint.
5. At a breakpoint, you can complete the following actions:
   - Press <F5> to continue execution. Program execution continues until the next breakpoint or until completion.
   - Single-step through any line of code by selecting **Run»Step Over** or **Run»Step Into**.
   - Stop the program at a breakpoint by pressing <Ctrl-F12> or by selecting **Run» Terminate Execution**.
6. To remove the breakpoint from the program, click the red diamond.

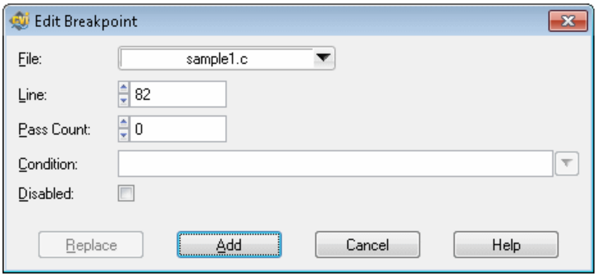
◆

Conditional Breakpoints

Use conditional breakpoints to halt program execution only when the specified condition is true. Complete the following steps to use conditional breakpoints in your program:

1. Select **Run»Breakpoints** to open the Breakpoints dialog box
2. In the Breakpoints dialog box, click **Add/Edit Item** to open the Edit Breakpoint dialog box.

Figure 4-4. Edit Breakpoint Dialog Box



Edit Breakpoint

File: sample1.c

Line: 82

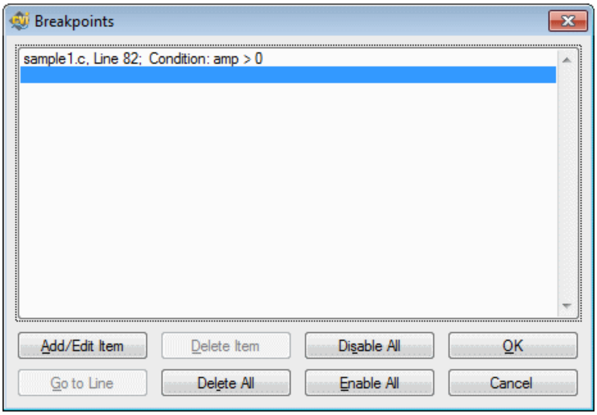
Pass Count: 0

Condition:

Disabled: ☐

3. In the Edit Breakpoint dialog box, enter `82` for **Line**, and enter `amp > 0` as the **Condition**. Notice the default values for the remaining controls, but do not change them. Click **Add**. The Breakpoints dialog box appears as shown in the following image.

Figure 4-5. Entering Breakpoint Information



Breakpoints

sample1.c, Line 82; Condition: amp > 0

4. Click **OK** to exit the Breakpoints dialog box. LabWindows/CVI displays a yellow square to the left of line 82 to indicate the conditional breakpoint.
5. Select **Run»Debug sample1.exe** to begin program execution. Click **Acquire** to run the code in the commit event case for `AcquireData`. LabWindows/CVI halts execution at line 82 because the breakpoint condition was met. Hover the mouse cursor over `amp` to verify its value is greater than 0.
6. Select **Run»Terminate Execution** to stop the program.

7. Right-click the conditional breakpoint icon to the left of line 82 and select **Breakpoints** to open the Breakpoints dialog box.
8. Click **Add/Edit Item** to open the Edit Breakpoint dialog box. Replace the **Condition** text with `amp < 0` and click **Replace**. Then click **OK** to exit the Breakpoints dialog box.
9. Repeat step 5. Notice that LabWindows/CVI does not halt execution at line 82 because the breakpoint condition is no longer true.
10. Press <Ctrl-F12> twice to stop the program. To remove the breakpoint, select **Run» Breakpoints**, ensure the breakpoint is highlighted, and click **Delete Item**. Then click **OK** to exit the dialog box.

Note For more information about breakpoints, refer to *breakpoints* in the *LabWindows/CVI Help* index.

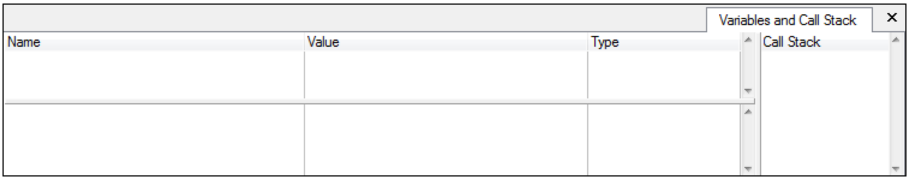
Displaying and Editing Data

Step mode execution and breakpoints are useful tools for high-level testing. However, you often need to look beyond your source code to test your programs. LabWindows/CVI provides displays for viewing and editing the data for your program. In the following exercises, you use a variety of these displays to view data generated by your application.

Variables and Call Stack Window

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

Figure 4-6. Variables and Call Stack Window

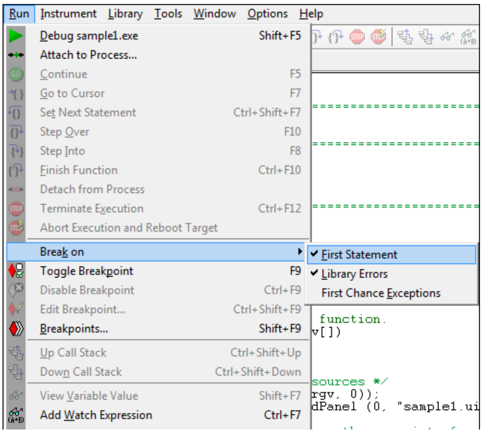


The Variables and Call Stack window lists the name, value, and type of currently active variables. LabWindows/CVI displays variables in categories according to how they are defined, such as global or local. The Call Stack section shows the current call stack of functions. To view variables that are active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

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

1. Select **Run»Break on»First Statement**, as follows.

Figure 4-7. Breaking On First Statement



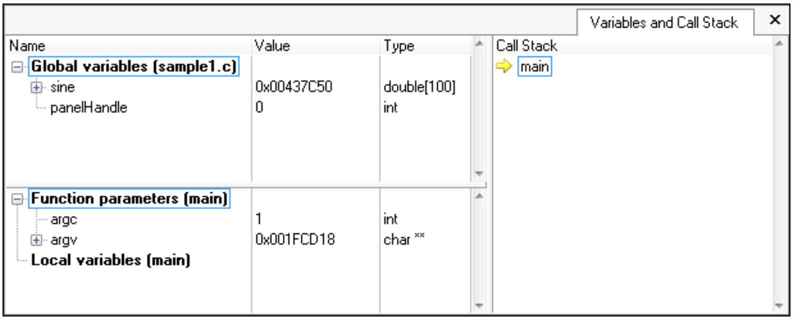
Run
Instrument
Library
Tools
Window
Options
Help
Debug sample1.exe Shift+F5
Attach to Process...
Continue F5
Go to Cursor F7
Set Next Statement Ctrl+Shift+F7
Step Over F10
Step Into F8
Finish Function Ctrl+F10
Detach from Process
Terminate Execution Ctrl+F12
Abort Execution and Reboot Target
Break on
First Statement
Library Errors
First Chance Exceptions
Toggle Breakpoint F9
Disable Breakpoint Ctrl+F9
Edit Breakpoint... Ctrl+Shift+F9
Breakpoints... Shift+F9
Up Call Stack Ctrl+Shift+Up
Down Call Stack Ctrl+Shift+Down
View Variable Value Shift+F7
Add Watch Expression Ctrl+F7

```
 function
v[])

sources */
rgv, 0));
dPanel (0, "sample1.ui
```

2. Select **Run»Debug sample1.exe** to run the program. When the program begins execution, LabWindows/CVI highlights the `main` function in the Source window.

3. Select **Window»Variables and Call Stack** to view the Variables and Call Stack window, shown in Figure 4-8.

Figure 4-8. Variables and Call Stack Window During Execution of main



Name	Value	Type
Global variables (sample1.c)		
sine	0x00437C50	double[100]
panelHandle	0	int
Function parameters (main)		
argc	1	int
argv	0x001FCD18	char **
Local variables (main)		

Note The values you see for your project might differ from the values shown in Figure 4-8.

4. Insert a breakpoint on the line with the following code:

   ```
   SinePattern (100, amp, 180.0, 2.0, sine);
   ```

5. Press <F5> to continue program execution. Click **Acquire**. LabWindows/CVI halts program execution on the statement with the breakpoint. In the Variables and Call Stack window, LabWindows/CVI now lists `AcquireData` in the Call Stack section. The Variables and Call Stack window shows the variables that are declared locally to that function in the **Local variables** section of the window.
6. Leave the program suspended and continue to the next section, *Editing Variables*.

Editing Variables

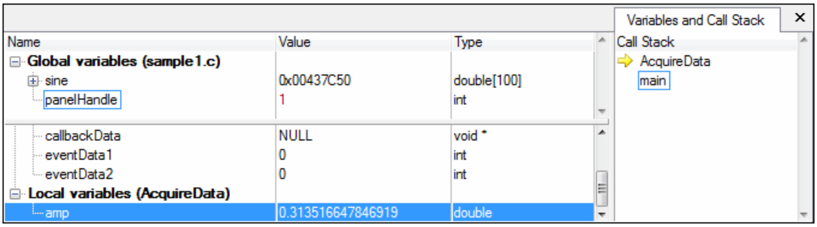
In addition to displaying variables, you can use the Variables and Call Stack window to edit the contents of a variable. Complete the following steps to use the Variables and Call Stack window for this purpose:

1. Make sure the `sample1.c` program is still suspended on the following line:

   ```
   SinePattern (100, amp, 180.0, 2.0, sine);
   ```

2. Highlight the `amp` variable in the Source window and select **Run»View Variable Value**. LabWindows/CVI highlights the `amp` variable in the Variables and Call Stack window.

Figure 4-9. Highlighting the `amp` variable



Variables and Call Stack

Name	Value	Type
Global variables (sample1.c)		
sine	0x00437C50	double[100]
panelHandle	1	int
callbackData	NULL	void *
eventData1	0	int
eventData2	0	int
Local variables (AcquireData)		
amp	0.313516647846919	double

Call Stack
- AcquireData
- main

3. From the Variables and Call Stack window, press <Enter> to edit the value of `amp`. Enter `0.2` in the value column and press <Enter>.
4. In the Source window, select **Run»Continue**. Notice that the sine pattern amplitude is now `0.2`. The change you made using the Variables and Call Stack window took effect immediately in the execution of the program.

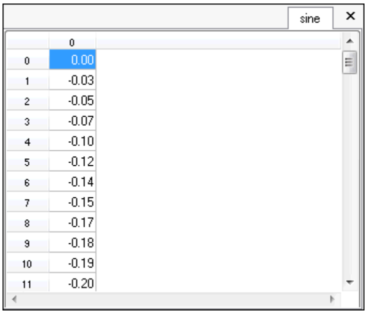
Note Notice that LabWindows/CVI displays in red text those variable values that changed since the program was last suspended. In the Variables and Call Stack window, LabWindows/CVI now displays the value for the `amp` variable in red text to indicate a changed value.

Array Display Window

The Array Display window shows the contents of an array of data or a string. You can use the Array Display window to edit array or string elements in the same way that you edited variables using the Variables and Call Stack window.

1. Click **Acquire** to put the program in breakpoint mode again.
2. Right-click `sine` in the Variables and Call Stack window and select **View»Array Display** to view the array values as shown in Figure 4-10.

Figure 4-10. Array Display Window



	0
0	0.00
1	-0.03
2	-0.05
3	-0.07
4	-0.10
5	-0.12
6	-0.14
7	-0.15
8	-0.17
9	-0.18
10	-0.19
11	-0.20

Note The actual values in your array might differ from the values shown in Figure 4-10.

The Array Display window shows the values of array elements in tabular format. In Figure 4-10, the `sine` 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.

Take a moment to view the display. You can edit individual elements in the array just as you edited variables in the Variables and Call Stack window.

3. Close the Array Display window.

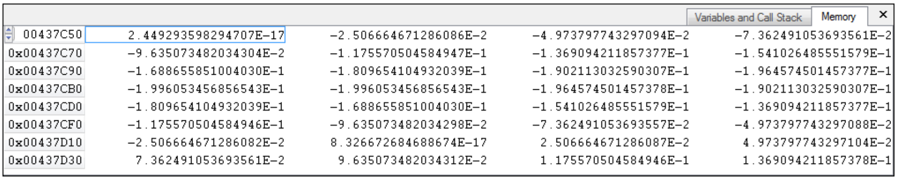
Memory Display Window

You can use the Memory Display window to view and edit the memory of the program you are debugging. Use the Memory Display window as follows:

1. With your program still suspended, select **Window»Memory** to display the Memory Display window.
2. Click the **Variables and Call Stack** tab to return to the Variables and Call Stack window.

3. Click the `sine` variable in the Variables and Call Stack window and drag it to the **Memory** tab. LabWindows/CVI displays the `sine` array memory in the Memory Display window, shown in Figure 4-11.

Figure 4-11. Memory Display Window



```
  00437C50   2.449293598294707E-17   -2.506664671286086E-2   -4.973797743297094E-2   -7.362491053693561E-2
0x00437C70   -9.635073482034304E-2   -1.175570504584947E-1   -1.369094211857377E-1   -1.541026485551579E-1
0x00437C90   -1.688655851004030E-1   -1.809654104932039E-1   -1.902113032590307E-1   -1.964574501457377E-1
0x00437CB0   -1.996053456856543E-1   -1.996053456856543E-1   -1.964574501457378E-1   -1.902113032590307E-1
0x00437CD0   -1.809654104932039E-1   -1.688655851004030E-1   -1.541026485551579E-1   -1.369094211857377E-1
0x00437CF0   -1.175570504584946E-1   -9.635073482034298E-2   -7.362491053693557E-2   -4.973797743297088E-2
0x00437D10   -2.506664671286082E-2    8.326672684688674E-17   2.506664671286087E-2    4.973797743297104E-2
0x00437D30    7.362491053693561E-2    9.635073482034312E-2   1.175570504584946E-1    1.369094211857378E-1
```

4. To edit the program memory, right-click in the Memory Display window and select **Edit Mode**. When the Memory Display window is in edit mode, double-click a value to edit it. Similar to the Variables and Call Stack window, the Memory Display window also displays changed values in red text.
5. When you are finished, click <Ctrl-F12> to terminate program execution.

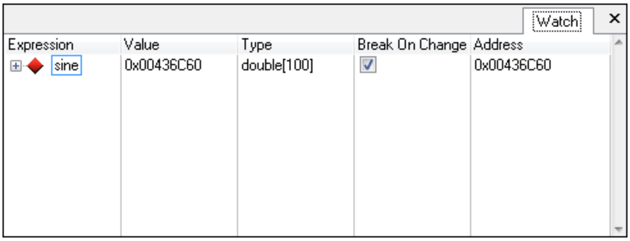
Watch Window

The Watch window is a powerful debugging tool. In addition to viewing values of variables changing dynamically as your program executes, you also can use the Watch window to view expression values and set conditional breakpoints when variable or expression values change. Complete the following steps to use the Watch window to view variables during program execution:

1. With `sample1.prj` still loaded as the current project, ensure that **Run»Break on» First Statement** is enabled. Click the breakpoint on the `SinePattern` line of code to remove it.
2. Select **Run»Debug sample1.exe** to start program execution. Execution breaks with the `main` function highlighted.
3. In the Variables and Call Stack window, right-click the `sine` variable and select **Add Watch Expression** to add the `sine` variable to the Watch window.

4. Enable the **Break On Change** option so that the window matches the one shown in Figure 4-12. You might need to resize the Watch window to expose the **Break On Change** option. Expand the `sine` variable within the Watch window to view the individual elements within the array.

Figure 4-12. Watch Window



Expression	Value	Type	Break On Change	Address
⊞ sine	0x00436C60	double[100]	✓	0x00436C60

5. Select **Run»Continue** to continue program execution.
6. Click **QUIT** on the user interface to exit the program. Remove the watch expression by clicking the `sine` variable in the Watch window and pressing <Delete>.

Tooltips

You also can use the following method to edit variables:

1. Disable the **Run»Break on»First Statement** option and set a breakpoint on the line of code that includes the `SinePattern` function call. Then, select **Run»Debug sample1.exe**.
2. Click **Acquire** on the user interface. Program execution breaks on the `SinePattern` statement.
3. Position the mouse cursor on the `amp` variable in the `SinePattern` statement.
4. The variable value appears in a tooltip. Highlight the current value and enter `3.0`.
5. Select **Run»Continue** to complete program execution. Notice the amplitude of the graphed sine pattern is the value you specified in the tooltip, not the amplitude calculated in the program. Click **QUIT** on the user interface to exit the program.

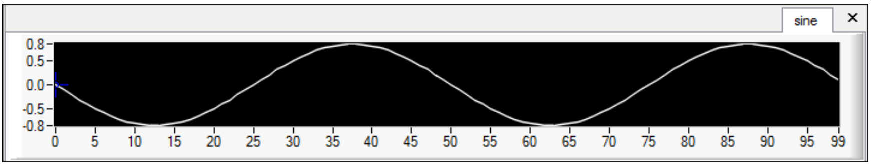
Graphical Array View

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

1. Clear the existing breakpoint. Then, set a breakpoint on the line of code that includes the call to `PlotY`.
2. Select **Run»Debug sample1.exe** and click **Acquire** on the user interface.

3. In the Variables and Call Stack window, highlight the `sine` variable and select **View»Graphical Array View** to view the `sine` values in a graph.

Figure 4-13. Graphical Array View



4. Select **Run»Continue** to complete program execution. Then click **QUIT** on the user interface to exit the program.

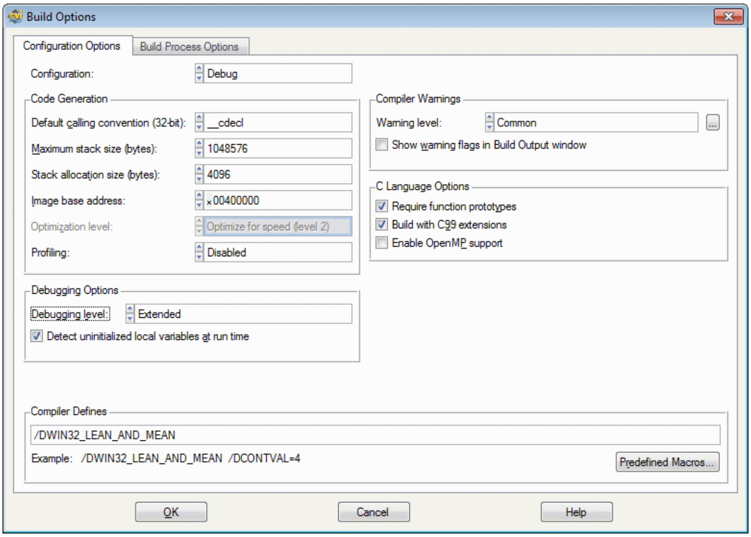
Resource Tracking Window

Use the Resource Tracking window to detect memory leaks in a LabWindows/CVI application. The Resource Tracking window displays resources that LabWindows/CVI has allocated, or recently deallocated, in the application. If the Resource Tracking window displays no resources when the program has completed execution, all allocated resources were subsequently deallocated. Resource tracking is enabled by default in extended debugging mode. Complete the following steps to view resources during program execution:

Note The Resource Tracking window is available only in the Full Development System.

1. Select **Options»Build Options** to open the Build Options dialog box. Ensure that the value in the **Debugging level** pull-down menu is **Extended**.

Figure 4-14. Build Options Dialog Box



Build Options

Configuration Options | Build Process Options

Configuration: Debug

Code Generation
Default calling convention (32-bit): __cdecl
Maximum stack size (bytes): 1048576
Stack allocation size (bytes): 4096
Image base address: 0x00400000
Optimization level: Optimize for speed (level 2)
Profiling: Disabled

Compiler Warnings
Warning level: Common
☐ Show warning flags in Build Output window

C Language Options
☑ Require function prototypes
☑ Build with C99 extensions
☐ Enable OpenMP support

Debugging Options
Debugging level: Extended
☑ Detect uninitialized local variables at run time

Compiler Defines
/DWIN32_LEAN_AND_MEAN
Example: /DWIN32_LEAN_AND_MEAN /DCONTVAL=4

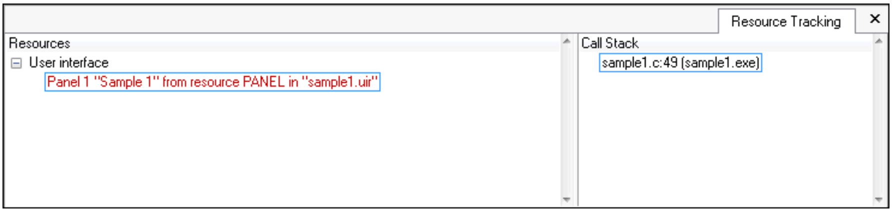
2. Click **OK** to exit the Build Options dialog box.
3. Verify the breakpoint still exists on the line of code that includes the call to `PlotY`. If not, add a breakpoint on that line of code.

Note The Resource Tracking window displays information only when the program is suspended. Therefore, the program must execute to the breakpoint before the window displays any information.

4. Select **Window»Resource Tracking** to open the Resource Tracking window.
5. Select **Run»Debug sample1.exe** to execute the program.
6. Click the **Acquire** button.

The Resource Tracking window appears similar to the following image.

Figure 4-15. Resource Tracking Window



Resource Tracking

Resources
⊟ User interface
Panel 1 "Sample 1" from resource PANEL in "sample1.uir"

Call Stack
sample1.c:49 (sample1.exe)

Notice the user interface resource for the panel that appears in the **Resources** column. LabWindows/CVI displays newly allocated resources in red text. Double-click the panel in the **Resources** column. LabWindows/CVI highlights the resource allocation of the panel in the Source window. The **Call Stack** column displays the call stack of functions when the resource was allocated.

Right-click on a resource in the Resource Tracking Window. The allocation is highlighted. Call the appropriate function to deallocate memory.

Where to Go Next

Refer to the **LabWindows/CVI Fundamentals»Debugging Tools»Resource Tracking Window** topic in the *LabWindows/CVI Help* for more information about the Resource Tracking window. When ready, proceed to the exercises in Chapter 5.

5 Adding Analysis to Your Program

This chapter builds on the concepts that you learned in Chapter 3, *Using Function Panels and Libraries*. If you did not complete the exercise in Chapter 3, go back and do so now.

In Chapter 3, *Using Function Panels and Libraries*, you generated code to plot the sine pattern array on the graph control. You placed the plotting function in a callback function that triggers by clicking the **Acquire** button. In this chapter, you add analysis code to write a callback function that finds the maximum and minimum values of the array and displays them in numeric indicators on the user interface.

Setting Up

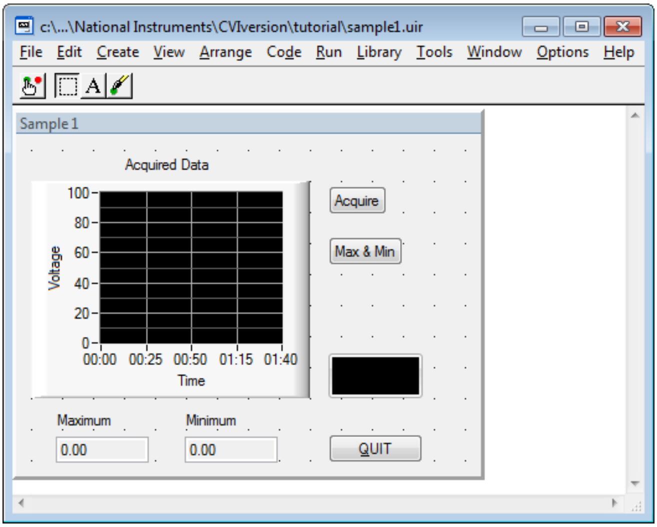
1. Remove all breakpoints and close all windows except the Workspace window.
2. Run `sample1.prj` to verify the operation of the program. Click **QUIT** to terminate the execution.

Modifying the User Interface

Complete the following steps to modify the existing user interface:

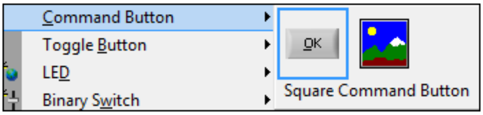
1. Open `sample1.c`. Place the cursor at the end of the file. CodeBuilder uses that location for the new callback function that it generates later in this chapter.
2. Without closing the `sample1.c` source code, open `sample1.uir`. Your goal is to modify the `.uir` to match the user interface shown in Figure 5-1.

Figure 5-1. Sample User Interface


c:\...\National Instruments\CVIversion\tutorial\sample1.uir
File Edit Create View Arrange Code Run Library Tools Window Options Help
Sample 1
Acquired Data
Voltage
100
80
60
40
20
0
00:00 00:25 00:50 01:15 01:40
Time
Acquire
Max & Min
Maximum
0.00
Minimum
0.00
QUIT

3. Add a command button to the panel.

Figure 5-2. Command Button



4. Select the new command button then enter the following information in the Attribute Browser.

Control	Value
Constant Name	`MAXMIN`
Callback Function	`FindMaxMin`
Label Text	`Max & Min`

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

Note The lightning bolt cursor appears while CodeBuilder generates code in the `sample1.c` source file. When you finish updating the user interface for Sample 1, you will add code to the `FindMaxMin` callback function to compute and display the maximum and minimum values of the array.

6. In the User Interface Editor, right-click the panel and select **Numeric»Numeric**.
7. Ensure the numeric control has focus and enter the following information in the Attribute Browser.

Control	Value
Constant Name	`MAX`
Control Mode	**indicator**
Label Text	`Maximum`

8. Add a second numeric control to the panel.
9. Ensure the numeric control has focus and enter the following information in the Attribute Browser.

Control	Value
Constant Name	`MIN`
Control Mode	**indicator**
Label Text	`Minimum`

10. Position the new controls on the user interface to match those shown in Figure 5-1.

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

11. Save the modified `.uir` file.

Writing the Callback Function

Now that you have modified the `.uir` file and generated the shell for the callback function for the **Max & Min** command button, you must complete the `FindMaxMin` function in the source file, as follows:

1. To quickly locate the `FindMaxMin` callback function in your source file, right-click the **Max & Min** button in the User Interface Editor and select **View Control Callback**. LabWindows/CVI displays the `sample1.c` source file with the `FindMaxMin` callback function highlighted.
2. Position the cursor on the blank line just after the `case EVENT_COMMIT:` statement.

Note LabWindows/CVI provides source code completion options within the Source window. You can use the **Edit»Show Completions** option to view a list of potential matches for functions or variables you are typing.

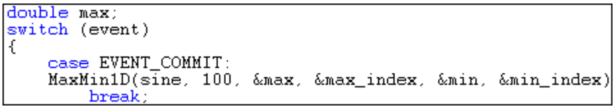
3. Type `Max` and then press <Ctrl-Space> to view the drop-down list of matches. Select **MaxMin1D** from the list.
4. Type an open parenthesis after the function name to display the function prototype. If you do not see the prototype after you type the parenthesis, press <Ctrl-Shift-Space>.

Note The prototype provides many of the same features as a function panel. As you type, LabWindows/CVI highlights the appropriate parameter name in the prototype tooltip. When you press <F1>, LabWindows/CVI displays help for the highlighted item.

5. `MaxMin1D` finds the maximum and minimum values of an array. Enter the following values for the parameters:

Parameter	Value
InputArray	`sine`
NumberofElements	`100`
MaximumValue	`&max`
MaximumIndex	`&max_index`
MinimumValue	`&min`
MinimumIndex	`&min_index`

6. Before you proceed, you must declare the `max`, `max_index`, `min`, and `min_index` variables. Place the cursor over the `max` variable name and press <Ctrl-D>. LabWindows/CVI inserts a copy of the `max` variable declaration at the top of the code block that contains your current position.



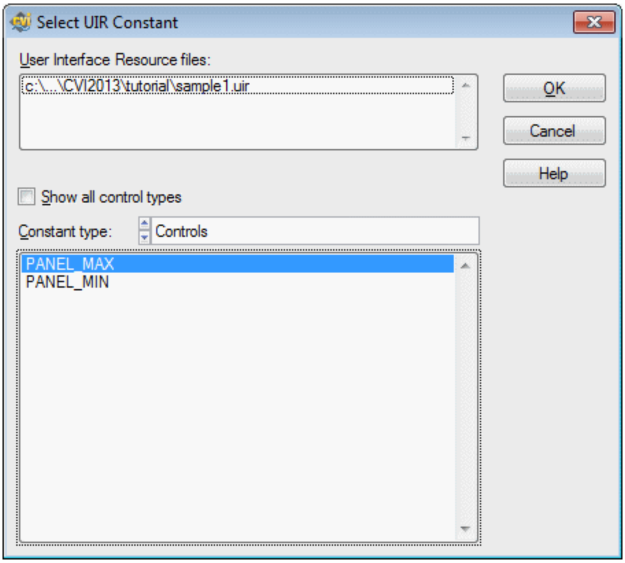
```
double max;
switch (event)
{
    case EVENT_COMMIT:
    MaxMin1D(sine, 100, &max, &max_index, &min, &min_index)
        break;
```

7. Repeat step 6 for the `max_index`, `min`, and `min_index` variables.
8. Enter a new line after the call to `MaxMin1D` and type `SetCtrlVal (` to display the function prototype for `SetCtrlVal`. If LabWindows/CVI does not display the prototype, press <Ctrl-Shift-Space> to view the tooltip.
9. The `SetCtrlVal` function sets the value of a control on your user interface. Enter `panelHandle` for the **PanelHandle** parameter. Then enter a comma to highlight the **ControlID** parameter in the prototype.
10. When **ControlID** is the highlighted parameter in the function prototype, LabWindows/CVI displays a **...** button next to the parameter name. This button indicates that

LabWindows/CVI provides an input selection dialog box or list of constant values for the current parameter.

11. Click this button or press <Ctrl-Shift-Enter> to launch the Select UIR Constant dialog box. Select **sample1.uir** in the **User Interface Resource files** list and select **PANEL_MAX** from the list of constants. Then click **OK**.

Figure 5-3. Select UIR Constant Dialog Box



c:\...\CVI2013\tutorial\sample1.uir

Constant type: Controls

PANEL_MAX
PANEL_MIN

12. Enter `max` for the value parameter, which is indicated by a ... in the function prototype.
13. On the next line, include another instance of `SetCtrlVal` with the following parameter values to set the value of the **Minimum** control on the user interface.

Parameter	Value
PanelHandle	`panelHandle`
ControlID	`PANEL_MIN`
Value (...)	`min`

14. Confirm that the source code matches the following code:

```
int CVICALLBACK FindMaxMin (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    ssize_t min_index;
    double min;
    ssize_t max_index;
```

```
        double max;
        switch (event)
        {
            case EVENT_COMMIT:
                MaxMin1D (sine, 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;
    }
```

Tip Notice the `ssize_t` data type in the previous example code. `ssize_t` and `size_t` are signed and unsigned integer data types, respectively, that do not rely on a specific pointer size. The size of `ssize_t` and `size_t` depend on the bitness of your application. When writing code to execute on both 32- and 64-bit systems, use data types that are not fixed-sized. Refer to the **Creating Applications»Creating 64-bit Applications Versus 32-bit Applications»Porting 32-bit Code to 64-bit Code** topic in the *LabWindows/CVI Help* for more information about writing code for 64-bit applications.

Running the Program

You have now successfully written the callback function. Save the files and run the project.

During program execution, the `FindMaxMin` function is called when you click **Max & Min**. When you click **Max & Min**, three separate events occur:

1. First, clicking the left mouse button generates an `EVENT_LEFT_CLICK` event.
2. Next, releasing the left mouse button generates an `EVENT_COMMIT` event. You wrote the function so that it finds the minimum and maximum values and displays them only when your program receives the `EVENT_COMMIT` event.
3. Finally, the button gets the input focus. For more practice with user interface events, complete *Exercise 4: Adding User Interface Events* of Chapter 7, *Additional Exercises*.

Where to Go Next

Stop debugging the application and proceed to Chapter 6, *Distributing Your Application*.

6

Distributing Your Application

This chapter describes how to distribute an application you create LabWindows/CVI. If you did not complete the tutorial exercises in Chapters 2 through 5, go back and do so now.

You can use the LabWindows/CVI distribution creation and management features to develop and edit multiple distributions for 32-bit or 64-bit applications. This tutorial includes steps for creating a basic 32-bit Windows Installer (`.msi`). The steps for creating a 64-bit installer are similar. For information about porting 32-bit code to 64-bit code and creating a 64-bit Windows Installer, refer to the **Creating Applications»Creating 64-bit Applications Versus 32-bit Applications** section in the *LabWindows/CVI Help*.

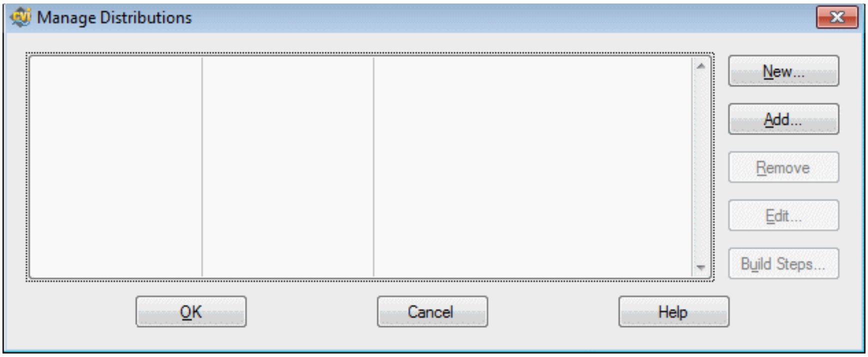
For more advanced information about distributing applications, refer to the **Distributing Applications»Creating and Editing an Installer** topic of the *LabWindows/CVI Help*.

Creating a New Distribution

Complete the following steps to create a new installer for your application:

1. Select **Build»Distributions»Manage Distributions** to open the Manage Distributions dialog box.

Figure 6-1. Manage Distributions Dialog Box



2. Click **New** to launch the New Distribution dialog box. Enter `Sample Distribution` as the **Name**. Verify the path in the **Settings file** field is the following:

   `\Users\Public\Documents\National Instruments\CVI`*`version`*`\tutorial\Sample1.cds`.

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

Editing the Distribution

When you create a new distribution, LabWindows/CVI launches the Edit Installer dialog box where you can specify various distribution components and features. Complete the following steps to verify and edit the distribution settings for your application:

1. In the **General** tab, verify the **Output directory**. LabWindows/CVI builds the installer in this location. Ensure that the **Output directory** is the following folder:

   `\Users\Public\Documents\National Instruments\CVI`*`version`*`\`
   `tutorial\cvidistkit.%name`.

2. Verify that the **Auto-increment** option is enabled. This option ensures that LabWindows/CVI increments the version number each time you build the installer.

   Note National Instruments recommends you install an upgrade installer—an installer that has a later version number than the previous installer—every time you install an application to a location where another version of that application might be installed. Upgrade installers uninstall the previous version of the application before installing the updated version.

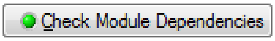
3. Click the **Files** tab. By default, LabWindows/CVI adds the project output (`sample1.exe`) and dependencies to the installation. These files are listed in the **Installation Files & Directories** section of the dialog box.

   Note Notice that Sample1 32-bit Output (`sample1.exe`) is listed in red text. Red text indicates that LabWindows/CVI cannot locate the file on your computer. In this case, you must build the release executable for LabWindows/CVI to include it in the installer. It is not necessary to exit the Edit Installer dialog box to build the executable. LabWindows/CVI builds the target automatically when it builds the distribution, as it does in step 9.

4. Click the **Shortcuts** tab. Notice that, by default, LabWindows/CVI includes a shortcut for `sample1.exe` in the `[Start»Programs]\Sample Distribution` directory.
5. Click the **Drivers & Components** tab. Notice that LabWindows/CVI includes the NI LabWindows/CVI Shared Runtime in the installer.

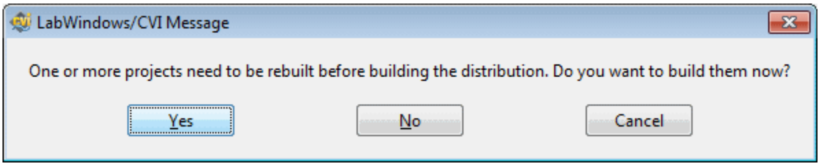
   Note Refer to the **Distributing Applications»Creating and Distributing Release Executables and DLLs»LabWindows/CVI Runtime»Side-By-Side Runtime** section of the *LabWindows/CVI Help* for more information about the shared and side-by-side runtime options.

6. Click **Check Module Dependencies** to ensure that any merge modules on which the selected drivers and components depend are included in the installer. LabWindows/CVI displays a message indicating that there are no missing dependencies and the LED on the button glows green.



7. (Optional) Click the **Registry Keys** and **Advanced** tabs to view the available options. For this application, it is not necessary to modify any of the settings in either of these tabs.
8. When you finish viewing and verifying the Edit Installer dialog box settings, click **OK** to exit the dialog box. Then click **OK** to exit the Manage Distributions dialog box.
9. Select **Build»Distributions»Build Sample Distribution**.
10. Click **Yes** in the message box LabWindows/CVI displays to prompt you to build the project. In some cases, LabWindows/CVI prompts you to insert the NI product installation media during the build.

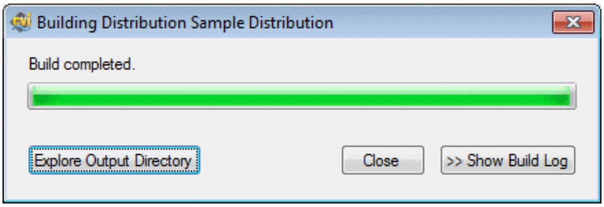
Figure 6-2. LabWindows/CVI Message


LabWindows/CVI Message

One or more projects need to be rebuilt before building the distribution. Do you want to build them now?

11. During the build process, LabWindows/CVI launches a dialog box that displays the build status. When LabWindows/CVI finishes building the installer, click **Close**. You are now ready to deploy the application to the target computer.

Figure 6-3. Build Status


Building Distribution Sample Distribution

Build completed.

Note Refer to the **Distributing Applications** section of the *LabWindows/CVI Help* for more information about managing distributions, as well as creating patches for existing distributions.

Deploying the Application to a Target Computer

In the previous steps, you built an installer that generated the files in the `\tutorial\cvidistkit.Sample Distribution` folder. In this exercise, there is only one folder, `Volume`. However, the folder can contain one or more `Volume` folders, the number of which depends on the size of the installer and the distribution media size. Once you create the installer, you can deploy it to a target computer.

Complete the following steps to copy the installer files and run the installer on the target computer:

1. Copy the `Volume` folder and its contents to the target computer. For this exercise, the target computer can be your development computer.
2. Double-click `setup.exe` to launch the installer for your application.
3. The installer displays a series of panels in which the user specifies the installation preferences. When the installer finishes updating the target system, click **Finish**.
4. To uninstall the application, use the **Programs and Features** option in the Windows Control Panel.

Note If you install the application to a computer other than your development computer, the installer includes the LabWindows/CVI Runtime and other National Instruments software necessary for your application, which you may want to remove.

5. To uninstall the NI LabWindows/CVI Runtime and other National Instruments software necessary for your application, use the **Programs and Features** option in the Windows Control Panel.
6. Select **National Instruments Software** from the list of currently installed programs and click **Uninstall/Change**. Select the NI products you want to remove and click **Remove**.

Where to Go Next

Proceed to the exercises in Chapter 7.

7 Additional Exercises

All of the exercises in this chapter build on the sample project that you completed in Chapter 5, *Adding Analysis to Your Program*. If you did not complete the sample project, go back and do so now. If you have trouble successfully completing the Chapter 5 exercise, start with `sample1.prj`.

This chapter provides additional exercises that build on the concepts you have used throughout this tutorial. Each exercise adds to the code that you develop in the preceding exercise. 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.

The solutions are located in the following folder:

`\Users\Public\Documents\National Instruments\CVI`*`version`*`\tutorial\solution`.

Exercise 1: Setting User Interface Attributes Programmatically

Each control on the `.uir` file has a number of control attributes that you can set to customize the look and feel of the control. When you build a user interface, you set the control attributes in the Attribute Browser or the Edit dialog boxes for the controls. For example, text font, size, and color are user interface control attributes.

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

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

Assignment

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

Tip To ensure multiple plots do not accumulate on the graph control, add a line of code to delete any existing plots before you call `PlotY`.

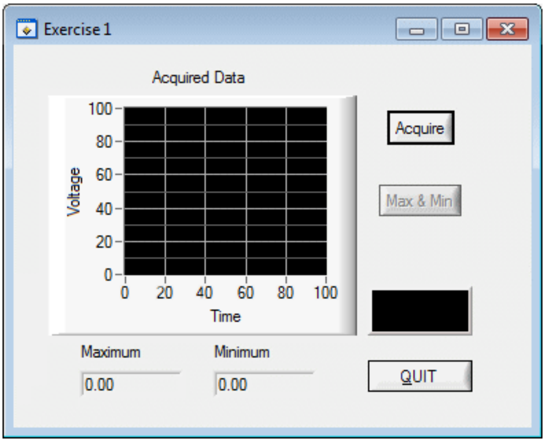
Hints

- Start by dimming the **Max & Min** command button in the User Interface Editor.
- Use `SetCtrlAttribute` from the User Interface Library to enable the **Max & Min** button.

Solution

Figure 7-1 displays the solution for this exercise. The solution for this exercise also can be found in the `\solution` folder under the filename `exer1.prj`.

Figure 7-1. Exercise 1 Solution



Exercise 2: Storing the Waveform on Disk

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

Tip When you must store very large data sets, National Instruments recommends that you use the TDM Streaming Library, which is optimized for handling large amounts of data.

Assignment

Use the file I/O functions in the ANSI C Library to save the `sine` array to a text file. Write the program so that the file is overwritten, not appended, each time you acquire the data.

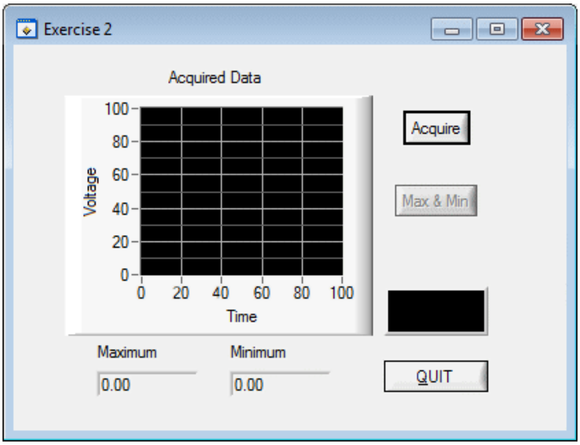
Hints

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

Solution

Figure 7-2 displays the solution for this exercise. The solution for this exercise also can be found in the `\solution` folder under the filename `exer2.prj`.

Figure 7-2. Exercise 2 Solution

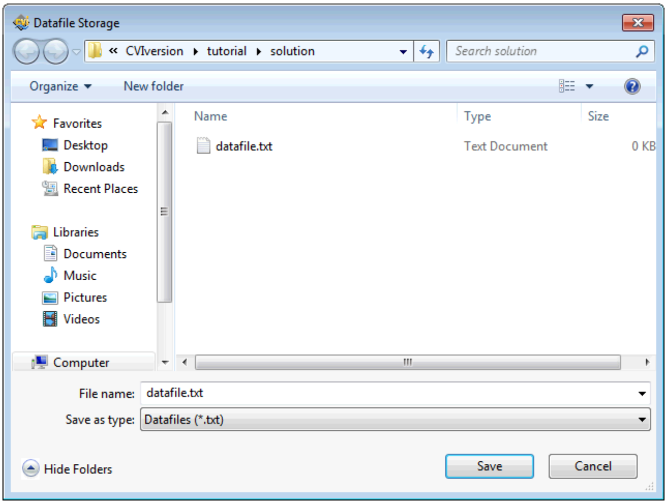


Exercise 3: Using Pop-Up Panels

The User Interface Library has a set of predefined panels called pop-up panels. Pop-up panels provide a quick and easy way to display information on the screen without developing a complete `.uir` file. You can use pop-up panels to prompt the user for input, confirm a selection, or display a message.

One of the most useful pop-up panels is generated with the `FileSelectPopupEx` function. With this pop-up panel, you can use a File Load or File Save dialog box, shown in Figure 7-3, to prompt the user to select or input a filename whenever your program must write to or read from a file.

Figure 7-3. Pop-Up Panel



Name	Type	Size
datafile.txt	Text Document	0 KB

File name: datafile.txt
Save as type: Datafiles (*.txt)

Assignment

Add a **Save** button to the `.uir` file so that the data in the array is saved only after the user clicks the **Save** button. When the user clicks the **Save** button, your program should launch a dialog box in which the user can define the drive, directory, and filename of the data file. When you finish, the `.uir` file should look similar to the one shown in Figure 7-4.

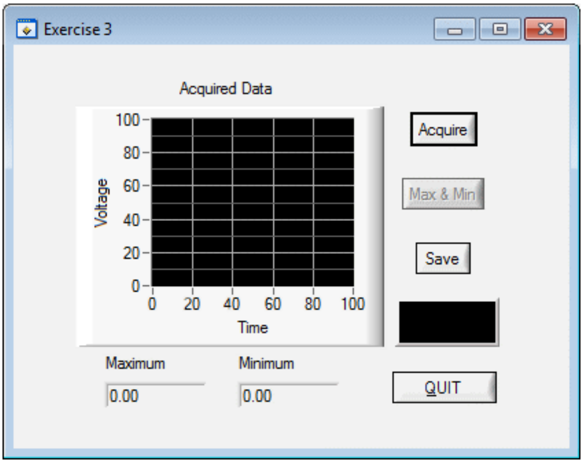
Hints

- When you create the **Save** button, assign a callback function to it.
- You must move the source code that you developed in Exercise 2 for writing the array to disk into the callback function.
- Before you write the data to disk, prompt the user for a filename with the `FileSelectPopupEx` function from the User Interface Library.

Solution

Figure 7-4 displays the solution for this exercise. The solution for this exercise can be found in the `\solution` folder under the filename `exer3.prj`.

Figure 7-4. Exercise 3 Solution



Exercise 4: Adding User Interface Events

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

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

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

To view the events that each user action generates, click the following icon, which puts the User Interface Editor into operate mode.

When the User Interface Editor is in operate mode, LabWindows/CVI displays events in the status bar located at the bottom of the Workspace window. Refer to the *Events Overview* topic in the *LabWindows/CVI Help* for a list of the events you can generate from a GUI.

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

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

Assignment

Many times, the person operating a LabWindows/CVI program is not the person who developed the program. The GUI might be very easy to use, but usually it is preferable to add help for the controls on `.uir` panels to assist the operator. Modify `exer3.prj` to display a short description for each command button when the user right-clicks the button.

Hints

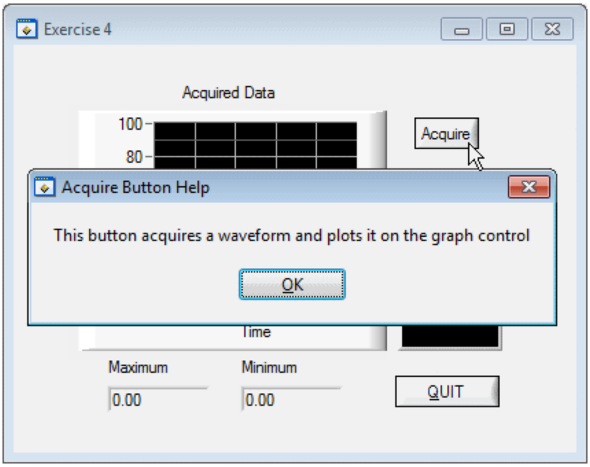
- Use `MessagePopup` to display the help.
- Remember that the event type is passed to each callback function in the event parameter.
- The event that you must respond to is `EVENT_RIGHT_CLICK`.

Tip If you want to add pop-up documentation to controls, use the `SetCtrlAttribute` function and specify text with the `ATTR_TOOLTIP_TEXT` attribute.

Solution

Figure 7-5 displays the solution for this exercise. The solution for this exercise can be found in the `\solution` folder under the filename `exer4.prj`.

Figure 7-5. Exercise 4 Solution



Acquire Button Help

This button acquires a waveform and plots it on the graph control

OK

Exercise 5: Timed Events

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

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

Assignment

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

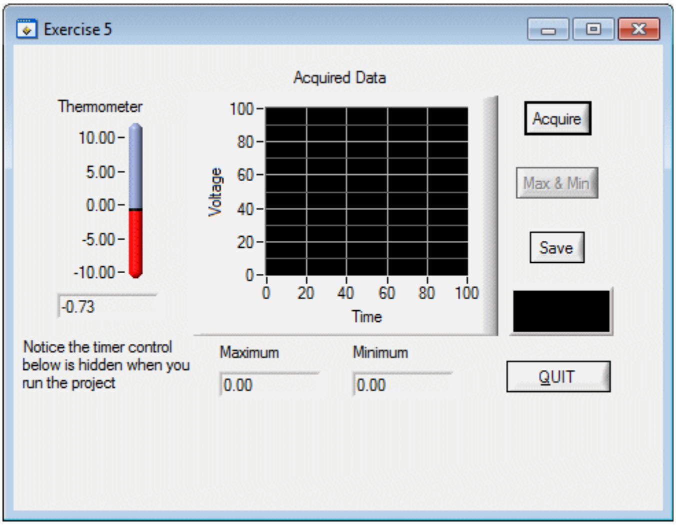
Hints

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

Solution

Figure 7-6 displays the solution for this exercise. The solution for this exercise also can be found in the `\solution` folder under the filename `exer5.prj`.

Figure 7-6. Exercise 5 Solution



Notice the timer control below is hidden when you run the project

8 Related Software Packages

NI offers additional packages for targeted applications with LabWindows/CVI. Visit the Product Catalog at ni.com for more information about the following software packages:

- **NI Developer Suite**—The NI Developer Suite includes the latest versions of LabVIEW, LabWindows/CVI, and Measurement Studio. You can customize the NI Developer Suite to include software options that suit your application needs. NI Developer Suite also includes a one-year subscription to the Standard Service Program, which provides quarterly software updates automatically and gives you one-on-one access to NI Applications Engineers for your technical support questions.
- **LabWindows/CVI Real-Time Module**—Create reliable and deterministic applications that target dedicated real-time hardware with this module that extends the LabWindows/CVI development environment. National Instruments provides a commercial off-the-shelf platform for real-time application development by combining flexible, high-performance software with rugged, modular hardware.
- **Real-Time Execution Trace Toolkit**—Create execution traces for LabWindows/CVI Real-Time applications with this toolkit. Interactively analyze and benchmark thread and function execution. Optimize performance by identifying memory allocation, sleep spans, and contention. Print trace sessions for documentation and code reviews and visually debug multicore applications.
- **LabWindows/CVI Execution Profiler Toolkit**—Acquire execution data to debug and optimize the execution speed of your LabWindows/CVI applications. The toolkit displays data in the Profile Viewer where you can open, browse, and analyze profiled data. Users with the LabWindows/CVI Full Development System can install the toolkit and activate it using the LabWindows/CVI Full Development System activation code. The toolkit can be purchased for use with the LabWindows/CVI Base Package.
- **TestStand**—TestStand is a ready-to-run test executive for organizing, controlling, and executing automated prototype, validation, or manufacturing test systems. TestStand is completely customizable, so you can modify and enhance it to match your specific needs. TestStand comes complete with integrated LabWindows/CVI tools.
- **NI Vision Development Module**—Use the NI Vision Development Module to develop machine vision and scientific imaging applications. The NI Vision Development module includes NI Vision, a library of powerful functions for image processing, and NI Vision Assistant, an interactive environment to quickly prototype vision applications without programming.
- **DIAdem**—DIAdem is an interactive tool for mathematical and visual data analysis, report generation, task automation, and data management. DIAdem imports data from files and industry-standard databases and can optimally handle datasets with more than one billion parts. Use the TDM Streaming Library to create TDMS files for use in DIAdem.

- **LabWindows/CVI SQL Toolkit**—The LabWindows/CVI SQL Toolkit provides a set of easy-to-use tools for quickly connecting to local and remote databases and implementing many common database operations without having to perform Structured Query Language (SQL) database operations. The toolkit readily connects to popular databases. If you need advanced database functionality and flexibility, the toolkit also offers complete SQL capabilities.
- **LabWindows/CVI Signal Processing Toolkit**—The LabWindows/CVI Signal Processing Toolkit provides tools for digital filter design, joint time-frequency analysis, wavelet and filter bank design, and super-resolution spectral analysis.
- **PID Control Toolkit**—The PID Control Toolkit adds sophisticated control algorithms to LabWindows/CVI. With this package, you can build data acquisition and control systems for your own control application.
- **Modulation Toolkit**—Use this toolkit to generate and analyze analog and digital modulated signals. You can use this toolkit to build applications that measure signal impairments, bit error rate, burst timing, phase noise, carrier frequency shift, modulation index, and complementary cumulative distribution function (CCDF) values of signals generated by a unit under test.
- **LabWindows/CVI Run-Time Module for Linux**—Create high-performance, stable applications on a Windows system and later compile and run these applications on a Linux® target.

A NI Services

National Instruments provides global services and support as part of our commitment to your success. Take advantage of product services in addition to training and certification programs that meet your needs during each phase of the application life cycle; from planning and development through deployment and ongoing maintenance.

To get started, register your product at `ni.com/myproducts`.

As a registered NI product user, you are entitled to the following benefits:

- Access to applicable product services.
- Easier product management with an online account.
- Receive critical part notifications, software updates, and service expirations.

Log in to your National Instruments `ni.com` User Profile to get personalized access to your services.

Services and Resources

- **Maintenance and Hardware Services**—NI helps you identify your systems' accuracy and reliability requirements and provides warranty, sparing, and calibration services to help you maintain accuracy and minimize downtime over the life of your system. Visit `ni.com/services` for more information.
  - **Warranty and Repair**—All NI hardware features a one-year standard warranty that is extendable up to five years. NI offers repair services performed in a timely manner by highly trained factory technicians using only original parts at a National Instruments service center.
  - **Calibration**—Through regular calibration, you can quantify and improve the measurement performance of an instrument. NI provides state-of-the-art calibration services. If your product supports calibration, you can obtain the calibration certificate for your product at `ni.com/calibration`.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

- **Training and Certification**—The NI training and certification program is the most effective way to increase application development proficiency and productivity. Visit `ni.com/training` for more information.
  - The Skills Guide assists you in identifying the proficiency requirements of your current application and gives you options for obtaining those skills consistent with your time and budget constraints and personal learning preferences. Visit `ni.com/skills-guide` to see these custom paths.
  - NI offers courses in several languages and formats including instructor-led classes at facilities worldwide, courses on-site at your facility, and online courses to serve your individual needs.
- **Technical Support**—Support at `ni.com/support` includes the following resources:
  - **Self-Help Technical Resources**—Visit `ni.com/support` for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at `ni.com/forums`. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Software Support Service Membership**—The Standard Service Program (SSP) is a renewable one-year subscription included with almost every NI software product, including NI Developer Suite. This program entitles members to direct access to NI Applications Engineers through phone and email for one-to-one technical support, as well as exclusive access to online training modules at `ni.com/self-paced-training`. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit `ni.com/ssp` for more information.
- **Declaration of Conformity (DoC)**—A DoC is our claim of compliance with the Council of the European Communities using the manufacturer's declaration of conformity. This system affords the user protection for electromagnetic compatibility (EMC) and product safety. You can obtain the DoC for your product by visiting `ni.com/certification`.

For information about other technical support options in your area, visit `ni.com/services`, or contact your local office at `ni.com/contact`.

You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index