LabWindows™/CVI™

User Manual

Worldwide Technical Support and Product Information

`ni.com`

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530,
China 86 21 6555 7838, Czech Republic 420 2 2423 5774, Denmark 45 45 76 26 00,
Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427,
India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970,
Korea 82 02 3451 3400, Malaysia 603 9131 0918, Mexico 001 800 010 0793, Netherlands 31 0 348 433 466,
New Zealand 1800 300 800, Norway 47 0 66 90 76 60, Poland 48 0 22 3390 150, Portugal 351 210 311 210,
Russia 7 095 238 7139, Singapore 65 6226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227,
Thailand 662 992 7519, United Kingdom 44 0 1635 523545

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

Important Information

Warranty

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

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

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

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

Copyright

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

Trademarks

CodeBuilder™, CVI™, DataSocket™, IVI™, LabVIEW™, National Instruments™, NI™, NI-488.2™, NI-CAN™, ni.com™, NI-DAQ™, NI Developer Zone™, and NI-VISA™ are trademarks of National Instruments Corporation.

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

Patents

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

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

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

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

Chapter 3
User Interface Editor

Chapter 4
Source and Interactive Execution Windows

Chapter 5
Using Instrument Drivers

Chapter 6
Using Function Panels

Chapter 7
Function Tree Editor

Chapter 8
Function Panel Editor

Chapter 9
Adding Help to Instrument Drivers

Chapter 10
Variables and Watch Windows

Chapter 11
Array and String Display Windows

Appendix A
Configuring LabWindows/CVI

Appendix B
Technical Support and Professional Services

Glossary

Index

About This Manual

The *LabWindows/CVI User Manual* contains detailed descriptions of LabWindows™/CVI™ features and functionality. To use this manual effectively, you should be familiar with the *Getting Started with LabWindows/CVI* manual, DOS, and Windows fundamentals.

Conventions

The following conventions appear in this manual:

[ ]	Square brackets enclose optional items—for example, [`response`].
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
	This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

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

- *LabWindows/CVI Help*
- *LabWindows/CVI Bookshelf*
- *LabWindows/CVI Quick Reference*
- *Getting Started with LabWindows/CVI*
- *LabWindows/CVI Programmer Reference Manual*
- *LabWindows/CVI Instrument Driver Developers Guide*
- *NI-488.2 Help*
- *Traditional NI-DAQ Function Reference Help*
- *Traditional NI-DAQ User Manual*
- *NI-DAQmx Help*
- *NI-VISA Programmer Reference Help*
- *National Instruments Example Finder Help*

1

Components of LabWindows/CVI

LabWindows/CVI is a programming environment for developing instrument control, automated test, and data acquisition applications in ANSI C.

LabWindows/CVI provides the following features:

- Standard libraries and interactive function panels for the following components:
    - GPIB
    - RS-232
    - Virtual Instrument Software Architecture (VISA)
    - Data acquisition
    - Data analysis
    - Transport Control Protocol (TCP)
    - DataSocket
    - Windows Dynamic Data Exchange (DDE) communication
    - File I/O
    - Data formatting
    - ANSI C
    - Analysis (or Advanced Analysis, available in the Full Development System)
- A graphical User Interface Editor, CodeBuilder wizard, and library for building, displaying, and controlling a graphical user interface
- A wizard and library for controlling ActiveX servers
- A wizard and library for creating IVI instrument drivers, which are highly structured VXI*plug&play*-compatible instrument drivers that use an attribute model to enable advanced features, such as state-caching, simulation, and compatibility with generic instrument classes
- A set of instrument drivers that contains high-level functions and interactive function panels for controlling specific instruments
- A tool for creating and editing NI-DAQmx tasks
- A tool for creating instrument control tasks

- A development environment with windows to manage projects and source code with complete editing, debugging, and user protection features

Standard Libraries

LabWindows/CVI includes the following standard libraries:

- User Interface Library
- Analysis Library/Advanced Analysis Library
- RS-232 Library
- NI-DAQmx Library
- Traditional NI-DAQ Library
- GPIB/GPIB 488.2 Library
- VISA Library
- IVI Library
- TCP Support Library
- DataSocket Library
- DDE Support Library
- ActiveX Library
- Formatting and I/O Library
- Utility Library
- ANSI C Library

LabWindows/CVI Environment

LabWindows/CVI provides an environment to create and test applications that use the LabWindows/CVI libraries. The environment is a combination editor, compiler, and debugger with extensive run-time checking. LabWindows/CVI also includes *function panels*, which make the task of developing programs much easier. Using a function panel, you can execute a LabWindows/CVI library function interactively and generate code that calls the function. Function panels also contain help for the functions and function parameters. You can build, execute, test, and debug the source code for your application in the LabWindows/CVI environment.

The LabWindows/CVI environment also has a User Interface Editor for creating a graphical user interface for your application programs. You can control the user interface using functions in the User Interface Library.

Also, you can use the LabWindows/CVI environment to create instrument drivers.

The LabWindows/CVI environment has the following windows, each with its own menu bar.

- **Workspace window**—This window appears when you start LabWindows/CVI. Use the Workspace window to open, edit, build, run, and save application project (`.prj`) files and to open, edit, and save workspace (`.cws`) files. A project contains a list of files your application uses. A workspace file contains the settings that do not affect the way a project builds. Workspaces can contain multiple projects.

- **User Interface Editor**—Use the User Interface Editor to build graphical user interfaces with pull-down menus, dialog boxes, controls, graphs, and strip charts and save them to user interface resource (`.uir`) files.

- **Source window**—Use Source windows to create, edit, run, debug, and save source code.

- **Interactive Execution window**—Use the Interactive Execution window to execute selected portions of code. You do not need to have a complete program in the Interactive Execution window, as is the case in a Source window. For instance, you can execute variable declarations and assignment statements in C without declaring a `main` function.

- **Function panels**—Use function panels to interactively execute library functions and insert code into a Source window.

- **Variables, Array Display, String Display, and Watch windows**—Use these windows to debug programs. You can view values of program variables in the Variables and Watch windows. The Array Display and String Display windows show contents of arrays and string variables at breakpoints.

- **Function Tree Editor**—Use the Function Tree Editor to build the tree structure of function panel files.

- **Function Panel Editor windows**—Use Function Panel Editor windows to build function panels.

- **Function Tree Help Editor and Function Panel Help Editor windows**—Use the Help Editor windows to add help to function panels, instruments, and classes.

You develop applications in the LabWindows/CVI environment using the ANSI C programming language. LabWindows/CVI complies with the ANSI X3.159-1989 and the ISO/IEC 9899:1990 standards for the C programming language.

You also can use LabWindows/CVI libraries with other compilers and linkers.

How to Create Applications with LabWindows/CVI

Use LabWindows/CVI as a text editor in which to enter your entire program. You can simplify application development by using function panels to execute LabWindows/CVI functions and to automatically insert the code into your program. Function panels contain complete online help.

A project contains all the component files of your application. The simplest case is one source file.

A typical project, however, contains multiple code modules and a user interface resource (`.uir`) file. You can include code modules as source files or compiled files. You can debug source files, and LabWindows/CVI performs run-time error checking when you execute code in source files.

To include compiled files such as library or object files in your project, you must compile them with LabWindows/CVI or a compatible external compiler. Compiled files consume less memory and run faster than source files. However, you cannot debug them, and they do not have run-time error checking.

You can mark a source file in the Project Tree to be compiled without debugging to use less memory.

You can strike a balance between initial project start-up time, execution speed, memory consumption, and the ability to debug code modules by varying the types of code modules you include in your project.

Creating a User Interface

You can create user interface objects (panels, controls, menus) using the User Interface Editor and save them in a `.uir` file. You can load, display, and modify these objects in your program using the functions in the User Interface Library. Also, you can specify callback functions that LabWindows/CVI calls when events occur on these objects.

LabWindows/CVI CodeBuilder automatically generates complete C code that compiles and runs based on a `.uir` file you create or edit. By choosing certain options presented to you in the **Code** menu, you can produce skeleton code. Skeleton code is syntactically and programmatically correct code that can compile and run before you type a single line of code. With the CodeBuilder feature, you save the time of typing standard code you must include in every program, eliminate syntax and typing errors, and maintain an organized source code file with a consistent programming style.

Creating Programs and DLLs

With the LabWindows/CVI Run-Time Engine, you can create executables, dynamic link libraries, and static libraries. Refer to Chapter 4, *Creating and Distributing Release Executables and DLLs*, of the *LabWindows/CVI Programmer Reference Manual* for more information.

2

Workspace Window

Use the Workspace window to open, edit, build, run, and save application project (`.prj`) files and to open, edit, and save workspace (`.cws`) files. A workspace file contains the settings that do not affect the way a project builds, such as breakpoints, window positions, tag information, and debugging levels. A workspace can contain one or more projects. A project file is a list of files your application uses. If you loaded a workspace the last time you used LabWindows/CVI, that workspace appears in the Workspace window when you start LabWindows/CVI again.

Workspace Window Overview

The Workspace window contains many of the different components of the LabWindows/CVI environment. In this window, you can work with all of your files—you can create, edit, save, and run files. You also can view output such as errors and search results.

The Workspace window contains the following four areas.
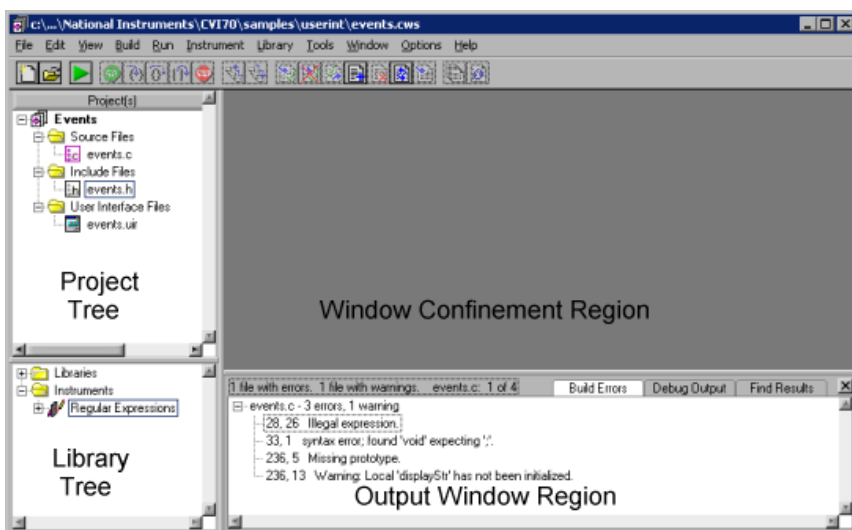
- **Project Tree**—Contains the list of files in each project included in your workspace. A bold project name indicates that project is the active project. The active project determines which project you can build, debug, modify, and so on. An asterisk next to the project name indicates that the project has been modified and needs to be saved.

  To open and edit files, you can double-click the filename. For `.fp` files, double-click the filename to open the Select Function Panel dialog box; double-click the image to open the `.fp` in the Function Tree Editor. For files not native to LabWindows/CVI, double-click the filename to open the file in the associated editor; double-click the image to attempt to open the file in a LabWindows/CVI Source window.

  If you attach the project to a source code control system, a box appears next to the icon for each file in the Project Tree. An empty box means that the file is under the source code control. A box with a red checkmark means that you currently have the file checked out. A box with a blue checkmark means that someone else has the file checked out.

  You also can view additional columns of information in the Project Tree. The columns include status information, the date and time the file was last modified, and the size of the file.

  Right-click a project name to perform the following actions:

  – Save the project.
  – Set the selected project as the active project.
  – Build the project.
  – Edit the project.
  – Remove the selected project from the workspace.
  – Print files in the project.
  – Add a folder to the project.
  – Add a file to the project.
  – Create a new file in the project.
  – Get the latest version of the project from source control.
  – Check out the project from source control.
  – Check in the project to source control.
  – Browse the project in the Source Code Browser.
  – Find items within the Project Tree.
  – Expand all items within the Project Tree.
  – Collapse all items within the Project Tree.

Right-click a file in a project, such as a source file or user interface file, to perform the following actions:

– Open the file in its window.

– Edit the file. This option applies only to `.fp` files and files that are not native to LabWindows/CVI.

– Save the file.

– Exclude the file from the build. This option applies only to source files.

– Enable the 'O' option. This option compiles a source file without debugging information.

– Replace the file with another file.

– Remove the file from the project.

– Compile the file. This option applies only to source files.

– Mark the source file for compilation. This option recompiles the file during the next project build.

– Get the latest version of the file from source control.

– Check out the file from source control.

– Check in the file to source control.

– Browse the file in the Source Code Browser. This option applies only to source files.

– Find an item within the Project Tree.

– Expand all items within the Project Tree.

– Collapse all items within the Project Tree.

Right-click a folder to perform the following actions:

– Rename the folder.

– Remove the folder.

– Add a file to the project.

– Create a new file in the project.

– Find an item within the Project Tree.

– Expand all items within the Project Tree.

– Collapse all items within the Project Tree.

Right-click an empty area of the Project Tree to perform the following actions:

– Sort files by name, pathname, file extension, and date.

– View the full pathnames of the files.

– Find an item within the Project Tree.

- – Expand all items within the Project Tree.

  – Collapse all items within the Project Tree.

- **Library Tree**—Contains a tree view of the functions in LabWindows/CVI libraries and instruments. When you load an instrument, the Instruments folder contains a tree view of the function panels for the instrument. Double-click the instrument in the Library Tree to open the Edit Instrument dialog box.

  To open a LabWindows/CVI library function panel, double-click the function name in the Library Tree.

  The Library Tree context menu contains the following options:

  – View library functions in alphabetical order, by function name or function panel title, or in a flat list instead of a hierarchical class organization.

  – Customize the Library Tree.

  – Open the function panel. This option applies only to functions.

  – Access help for libraries, classes, and functions.

  – Browse the function in the Source Code Browser. This option applies only to functions.

  – Find a library, class, or function within the Library Tree.

  – Expand all items within the Library Tree.

  – Collapse all items within the Library Tree.

- **Window Confinement Region**—Contains open Source windows, User Interface Editor windows, and Function Tree Editor windows. When you open any of these windows, the menu and toolbar change to reflect new options in those windows. You can release windows from the Window Confinement Region by selecting **Window»Release Window**.

- **Output Window Region**—Contains the Build Errors, Run-Time Errors, Source Code Control Errors, Debug Output, and Find Results windows. These windows contain lists of errors, output, and search matches. You can double-click items in the list to highlight that line in a file; you cannot double-click items for source code control errors.

  To release these windows from the Output Window Region, right-click in a window and select **Release Window**. Except for the Debug Output window, all of these windows automatically open in the Output Window Region when they are modified, unless you have released the windows. To view these windows, select the window you want to view from the **Window** menu. Use the titled tabs to move among various windows.

Your project must contain one or more of the following files, unless you use the files as instrument driver program files or load them dynamically using `LoadExternalModule`.

- Source files (`.c`)
- Object files (`.obj`)
- Library files (`.lib`)—DLL import libraries are in this category.

The following files are optional in your project file list.

- **Header (`.h`) files**—Listing `.h` files makes it easy to open them for viewing or editing and ensures that the compiler can find them.

- **User interface resource (`.uir`) files**—Listing `.uir` files makes it easy to open them for viewing or editing and ensures that LabWindows/CVI can find them when you debug your application.

- **Instrument driver function panel (`.fp`) files**—Listing `.fp` files lets LabWindows/CVI automatically load instruments when you open the project.

- **Instrument driver program files**—Listing these files overrides the loading precedence for instrument driver program files.

Opening and Loading Files, Projects, and Workspaces

You can open `.c`, `.h`, and `.uir` files in a workspace by double-clicking the filename directly. Double-clicking a `.fp` file opens the Select Function Panel dialog box for the instrument driver. You also can open files by dropping them onto the Window Confinement Region. Dropping a `.fp` file onto the Window Confinement Region opens the Function Tree Editor and loads the `.fp` file.

To add a file to the active project, drop the file onto the Project Tree.

When you drop a project file onto the Window Confinement Region, LabWindows/CVI loads the project in a new workspace. When you drop a project onto the Project Tree, LabWindows/CVI adds the project to the currently loaded workspace.

When you drop a workspace file onto the Window Confinement Region or Project Tree, LabWindows/CVI loads the workspace.

File Menu for the Workspace Window

This section explains how to use the commands in the **File** menu for the Workspace window.

File»New

Use the **New** command to open new, empty windows.

If you choose **Source** or **Include**, a new Source window appears in which you can create a new `.c` or `.h` file.

If you choose **User Interface**, a new User Interface Editor window appears in which you can create a new `.uir` file.

If you choose **Project**, the New Project Options dialog box appears. To include the new project in the existing workspace, select **Create Project in Current Workspace**. To add the project to a new workspace, select **Create Project in New Workspace**. LabWindows/CVI stores in the workspace the settings that do not affect the way your project builds. If you do not add the new project to the existing workspace, LabWindows/CVI creates a new workspace for the new project. You also are prompted to transfer project options—**Build Options**, **Include Paths**, **Command Line**, **Source Code Control Options**—from the old project to the new project.

If you choose **Workspace**, a dialog box appears with a message that asks if you want to unload the current workspace and its associated project(s). Select **Yes** to unload the current workspace. LabWindows/CVI also prompts you to save any changes to the currently loaded files.

If you choose **Function Tree**, a new Function Tree Editor window appears in which you can create a new `.fp` file.

File»Open

Use the **Open** command to open various types of files. When you select **Open**, a dialog box appears, prompting you for a filename to load. The **Directory History** ring displays a list of directories from which you have opened files previously.

If you choose **Source** or **Include**, a Source window appears with your specified `.c` or `.h` file.

If you choose **User Interface**, the User Interface Editor appears with your specified `.uir` file.

If you choose **Project**, LabWindows/CVI prompts you to save any modified files in the old project and then loads the specified project in the Project Tree. If you open a project that does not have a workspace of the same name in the same folder, LabWindows/CVI creates a new workspace for the project with default options. NI recommends that you open workspaces instead of projects.

Choose the **Workspace** command to open an existing workspace from the list in the dialog box. LabWindows/CVI prompts you to save any changes to the currently loaded files.

If you choose **Function Tree**, the Function Tree Editor appears with the specified `.fp` file.

File»Save All

The **Save All** command saves all open files to disk.

File»Set Active Project

Use the **Set Active Project** command to switch to a different project in the current workspace. The active project determines which project you build, debug, and so on.

File»Save Project

Use the **Save** command to write the project (`.prj`) file to disk.

File»Save Project As

Use the **Save As** command to write the project file to disk using a new name you specify.

File»Save Workspace

Use the **Save Workspace** command to write the currently loaded workspace file to disk. Saving the workspace does not save changes you made to projects within the workspace.

File»Auto Save Workspace

If you enable the **Auto Save Workspace** command, LabWindows/CVI automatically saves workspace files and projects in the workspace. When you load a workspace, the **Auto Save Workspace** command is initially enabled unless the file is read-only. If you enable the command, LabWindows/CVI automatically saves the workspace file whenever the workspace contains significant new or modified information. If you disable this command, the workspace file is saved only in the following cases:

- When you select the **Save**, **Save As**, or **Save All** command from the **File** menu.
- When you unload the workspace or exit LabWindows/CVI. LabWindows/CVI prompts you to save the file in this case.

Notice that if you disable the **Auto Save Workspace** command, LabWindows/CVI does not save the workspace file when you start debugging or running a program, even if you set the **Save Changes before Debugging** option in the Environment dialog box to **Always** or **Ask**.

File»Most Recently Closed Files

For your reference, two lists appear in the **File** menu.

- A list of the four most recently closed files, other than workspace and project files
- A list of the four most recently closed workspace or project files

File»Exit LabWindows/CVI

Use the **Exit LabWindows/CVI** command to close the current LabWindows/CVI session. If you have modified any open files since you last saved them, LabWindows/CVI prompts you to save them.

Edit Menu for the Workspace Window

This section explains how to use the commands in the Workspace window **Edit** menu.

Edit»Workspace

LabWindows/CVI stores in the workspace the settings that do not affect the way your project builds. You can include multiple projects in a workspace. When you create a new project, a dialog box prompts you to add the new project in the existing workspace or to create a new workspace for the project.

The Edit Workspace dialog box displays the project files you included in your workspace. The Edit Workspace dialog box has the following options:

- **Project Files**—Displays a list of projects contained in the workspace.

- **Add**—Adds a project to the workspace. You can browse to the project you want to add.

- **Remove**—Removes a project from the workspace.

- **Move Up**—Moves the selected project up one line. This list determines the order in which projects appear in the Project Tree.

- **Move Down**—Moves the selected project down one line.

- **OK**—Accepts your changes and closes the dialog box.

- **Cancel**—Closes the dialog box without accepting any changes.

Edit»Project

Use the Edit Project dialog box to make changes to projects. The Edit Project dialog box has the following options:

- **Project Label**—Contains the name of the project. You can type a new name for the project. This name identifies the project in the Project Tree.

- **Project Files**—Contains a tree view of the files in the project.

- **Use Absolute Path for File**—Forces LabWindows/CVI to store the file path as an absolute path in the project file, rather than a path relative to the project when you load this project file from a different location.

- **Add**—Opens the Add Files to Project dialog box, where you can select one or more files to add to the project.

- **Replace**—Opens the Replace File in Project dialog box, where you can enter or browse to a new file to use in place of the original file.

- **Remove**—Deletes the file from the project list.

- **Include Paths**—Opens the Include Paths dialog box, where you can specify the paths that LabWindows/CVI should use when searching for header files.

- **Source Code Control**—Opens the Source Code Control Options dialog box, where you can specify source code control settings.

- **OK**—Accepts changes made to the project and closes the dialog box.

- **Cancel**—Closes the dialog box without making any changes to the project.

Edit»Add Files to Project

Use the **Add Files to Project** command to add any type of file to the active project list. Choose any one of the file types listed in the menu to open the Add Files to Project dialog box and then select a file from the dialog box.

Use the **Source**, **Object**, and **Library** commands to add code modules to your project.

An import library (`.lib`) file must accompany each DLL. If you want to use a DLL in your project, you must include the import library rather than the DLL.

For more information about using DLLs in LabWindows/CVI, refer to Chapter 3, *Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

Use the **Include** command to add header files to your project. Including header files in your project makes it easier to access header files.

Use the **User Interface** command to add `.uir` files to your project. You can include `.uir` files in your project to make it easier to access to the files.

Use the **Instrument** command to add instrument drivers to your project. Instrument drivers that LabWindows/CVI loads through the project remain in memory while the project is open.

Use the **All Files** command to add any file to your project.

View Menu for the Workspace Window

This section explains how to use the commands in the Workspace window **View** menu.

View»Project Tree

Use this command to display the Project Tree. The Project Tree contains all of the project files in the currently loaded workspace.

View»Library Tree

Use this command to display the Library Tree. You can access LabWindows/CVI library function panels and function panels of loaded instrument drivers through the Library Tree.

View»Toolbar

Use this command to toggle between viewing and not viewing the Workspace window toolbar.

View»Window Confinement Region

Use this command to display the Window Confinement Region. The Window Confinement Region contains open Source windows, User Interface Editor windows, and Function Tree Editor windows.

View»Columns

Use this command to display status, date and time, and size columns for files in the Project Tree.

The following icons appear in the Status column in the Workspace window.

S	The file has been modified since you last saved it.
C	You have modified the file since you last compiled it, or you manually marked it for compilation.
O	This symbol applies only to source (`.c`) files and indicates that you selected **Enable 'O' option**. If you enable this option, LabWindows/CVI compiles the source file without debugging information. You can use this option to reduce the amount of memory required to build a project.
I	The file is associated with a loaded instrument driver. If not present, the `.fp` file is not loaded into memory.
A	The `.fp` file is contained in the project and is attached to or associated with a program file.
U	The `.fp` file is contained in the project and is not attached to any program file.

Build Menu for the Workspace Window

This section explains how to use the commands in the Workspace window **Build** menu. Commands in the **Build** menu include commands used to build and link projects, mark projects for compilation, and create application files.

Build»Configuration

The **Configuration** item opens a submenu from which you select the active configuration for your project. Set the configuration to **Debug** when you want to debug your executable or DLL. Set the configuration to **Release** when you are ready to build a release version of your executable, DLL, or static library. If you choose **LabVIEW Real-Time Support** as the **Run-Time Support** in the Target Settings dialog box or if you build a static library, the **Debug** configuration is dimmed.

✎ **Note**   You can set the names of the target executable, DLL, or library files for each configuration using the **Target Settings** menu item.

When you select the **Release** item in the **Configuration** submenu, source modules execute faster, but you cannot set breakpoints or use the Variables window. Also, you have no protection from run-time memory errors such as using bad pointers, over-indexing arrays, passing incorrect array sizes, and so on.

Build»Create Debuggable Executable

The **Create Debuggable Executable** menu item appears if you select **Build»Configuration»Debug** and **Build»Target Type»Executable**.

Use this menu item to compile and build an executable with debugging information. Use the **Debugging level** in the Build Options dialog box to select the degree of debugging information you want LabWindows/CVI to generate for the executable. To debug the executable you create with this command, select **Run»Debug** in Workspace, Source, Variables, Watch, Array Display, and String Display windows.

You can specify the filename of the executable, as well as other executable settings, by selecting the **Target Settings** menu item. You can set other creation and build options in the Build Options dialog box.

Build»Create Debuggable Dynamic Link Library

The **Create Debuggable Dynamic Link Library** menu item appears if you select **Build»Configuration»Debug** and **Build»Target Type»Dynamic Link Library**.

Use this menu item to compile and build a DLL with debugging information. Use **Debugging level** in the Build Options dialog box to select the degree of debugging information you want LabWindows/CVI to generate for the DLL.

You can specify the filename of the DLL, as well as other DLL settings, by selecting the **Target Settings** menu item. You can set other creation and build options in the Build Options dialog box.

The **Create Debuggable DLL** command also generates a DLL import library for the DLL.

Debugging DLLs

When you select the **Build**»**Create Debuggable Dynamic Link Library** command, LabWindows/CVI includes debug code in your DLL and generates a `.cdb` file that contains a symbol table and source position information necessary for debugging. The `.cdb` file has the same pathname as the DLL.

In the LabWindows/CVI development environment, you can debug only DLLs you create in LabWindows/CVI with the **Create Debuggable Dynamic Link Library** command. Other development environments cannot debug DLLs you create in LabWindows/CVI.

The amount of debugging information included in the DLL and debug file depends on the value of the **Debugging level** option in the Build Options dialog box.

Build»Create Release Executable

The **Create Release Executable** menu item appears if you select **Build»Configuration»Release** and **Build»Target Type»Executable**.

Use this menu item to compile and build an executable without debugging information. This command ignores the value of **Debugging level** in the Build Options dialog box.

You can specify the filename of the executable, as well as other executable settings, by selecting the **Target Settings** menu item. You can set other creation and build options in the Build Options dialog box.

Build»Create Release Dynamic Link Library

The **Create Release Dynamic Link Library** menu item appears if you select **Build»Configuration»Release** and **Build»Target Type»Dynamic Link Library**.

Use this menu item to compile and build a DLL without debugging information. This command ignores the value of **Debugging level** in the Build Options dialog box.

You can specify the filename of the DLL, as well as other DLL settings, by selecting the **Target Settings** menu item. You can set other creation and build options in the Build Options dialog box.

This command also generates a DLL import library for the DLL.

Build»Create Static Library

The **Create Static Library** menu item appears if you select **Build»Target Type»Static Library**. When you do, the **Release** item always is checked.

Use this menu item to compile and build a static library without debugging information. This command ignores the value of **Debugging level** in the Build Options dialog box.

You can specify the filename of the static library, as well as other static library settings, by selecting the **Target Settings** menu item. You can set other creation and build options in the Build Options dialog box.

If you include a .lib file in a static library project, LabWindows/CVI includes all object modules from the .lib in the static library. This process differs from creating an executable or DLL, in which LabWindows/CVI includes only the .lib modules that other modules in the project reference. In addition, LabWindows/CVI reports an error if you attempt to build a static library when you have a DLL import library in your project.

Build»Mark Project for Compilation

You can force LabWindows/CVI to compile the source files in a project during the next build with the **Mark Project for Compilation** command. When LabWindows/CVI marks source files for compilation, a C appears next to the filename in the Project Tree if you have **View»Columns»Status** enabled. When you modify source files, LabWindows/CVI automatically marks the files for compilation.

Build»Batch Build

Use the Batch Build dialog box to build multiple projects.

- **Configurations**—Displays all the valid configurations for the projects in your workspace.
- **Check All Debug**—Selects all the debug configuration items in the list.
- **Check All Release**—Selects all the release configuration items in the list.
- **Check None**—Deselects all the items in the list.
- **Build**—Activates and builds each selected configuration of each project.
- **Rebuild**—Forces LabWindows/CVI to recompile all source files in each selected configuration of each project.
- **Cancel**—Cancels the operation and removes the Batch Build dialog box.

Build»Target Type

The **Target Type** item opens a submenu in which you select the target type for your project. The target type determines what type of file you create when you execute the **Build»Create** command. The name of the **Create** command changes depending on the target type and configuration you select. You can select from the following target types:

- Executable
- Dynamic Link Library
- Static Library

The **Executable** and **Static Library** options are dimmed if you select **LabVIEW Real-Time Only** as the **Run-Time Support** in the Target Settings dialog box.

When you select **Static Library**, the **Debug Project** command in the **Run** menu is dimmed. The **Configuration»Debug** command also is dimmed. If you select **Dynamic Link Library**, you can use the **Run»Specify External Process** command to specify an external program that uses the DLL. When you do this, the **Debug Project** command changes to **Debug *xxx*.exe**, where *xxx*.exe is the name of the program you specify.

Build»Target Settings

The **Target Settings** item opens the Target Settings dialog box. The Target Settings dialog box contains different controls depending on the item you check in **Target Type** submenu.

Target Settings for Executables

When you set the **Target Type** to **Executable** and select **Build»Target Settings**, the Target Settings dialog box has the following options:

- **Application File**—The names of the executable files for the debug and release versions of your program. Change the value of **Application File** to edit the filenames for the debug and release configurations. You can use the **Browse** button to select an existing filename.

- **Application Title**—A descriptive title for your program. This title appears in the **Start** menu of Windows if you create a distribution kit. This title also appears in the Windows Add/Remove Programs dialog box.

- **Application Icon File**—A file that contains a descriptive graphical icon for your program.

- **Icon**—The graphical representation of the **Application Icon File**. You can double-click **Icon** to browse for an icon file on disk. The sample program `samples\apps\iconedit\iconedit.exe` ships with LabWindows/CVI so that you can create your own icon files.

- **Run-Time Support**—The run-time support for your stand-alone executable. If you select the **Instrument Driver Only** item, your project does not link to the entire set of LabWindows/CVI libraries but instead links to a smaller set of functions.

  Stand-alone executables you create with the **Full Run-Time Engine** item selected use the LabWindows/CVI Run-time Engine DLL, `cvirte.dll`. Stand-alone executables you create with the **Instrument Driver Only** item selected use `instrsup.dll` instead of `cvirte.dll`.

  If you use a stand-alone compiler and want to use `instrsup.dll`, include `cvi70\extlib\instrsup.lib` in your external compiler project instead of `cvirt.lib` and `cvisupp.lib`. Remember that when you use an external compiler, you link to that compiler's ANSI C library.

  `instrsup.dll` contains functions from the following libraries:

  - Formatting and I/O Library (except `ArrayToFile` and `FileToArray`)
  - RS-232 Library (except `InstallComCallback`)
  - Utility Library (selected functions only)
  - ANSI C Library

  Your project also can link to the following libraries:

  - Analysis or Advanced Analysis Library
  - GPIB/GPIB 488.2 Library
  - VXI Library
  - VISA Library
  - IVI Library
  - NI-DAQmx Library
  - Traditional NI-DAQ Library—If your project files call the `Config_Alarm_Deadband`, `Config_ATrig_Event_Message`, `Config_DAQ_Event_Message`, `DIG_Change_Message_Config`, `Get_DAQ_Event`, or `Peek_DAQ_Event` functions in the Traditional NI-DAQ Library, you will get link errors when you build your program in LabWindows/CVI. The link errors occur because the preceding functions use the LabWindows/CVI User Interface Library internally, which is not available in the `instrsup` and `cvi_lvrt` run-time engine DLLs. Your project files can call any of the other functions in the Traditional NI-DAQ Library.

  If you use an external compiler and want to use any of these libraries, refer to Chapter 3, *Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

  If you use the **Create Distribution Kit** command on a project that you link for instrument driver support only, LabWindows/CVI automatically selects **Instrument Driver Only** as the **Run-Time Engine Support**.

The following Utility Library functions are in `instrsup.dll`.

– `Beep`

– `DateStr`

– `Delay`

– `SyncWait`

– `Timer`

– `TimeStr`

– `RoundRealToNearestInteger`

– `TruncateRealNumber`

– `InStandaloneExecutable`

– `CVIRTEHasBeenDetached`

`instrsup.dll` does not support the Standard Input/Output window. Functions such as `FmtOut` or `ScanIn` return errors when you use them with `instrsup.dll`.

All the functions in `instrsup.dll` are multithread safe.

- **Embed Project .UIRs**—Embeds `.uir` files into the executable. Enabling this option allows you to ship only the executable, instead of including the `.uir` files also. Make sure that you include all `.uir` files loaded through `LoadPanel` or `LoadPanelEx` in your project if you want to ship only one executable. If you pass an absolute filename to `LoadPanel` or `LoadPanelEx`, the function will always look on disk for the `.uir` file. If you pass a simple filename, `LoadPanel` or `LoadPanelEx` will look first for an embedded `.uir`.

- **Generate Map File**—Creates a memory map for the executable. This map lists the address of every function and variable and the name of the function or variable. This option is useful if your program crashes and provides an address where the failure occurred. You can view the map file to check which function or variable the crash occurred in.

- **Create Console Application**—If you disable this option, LabWindows/CVI creates your executable as a Windows GUI application. If you enable this option, LabWindows/CVI creates your executable as a console application. Console applications create a Windows console window (Command Prompt or MS-DOS Prompt) and set the standard I/O port to the console. Create a console application if you want to redirect the standard input or output of your program.

- **Register ActiveX Server After Build**—LabWindows/CVI registers the target each time that it builds the target. By default, this option is enabled for a new ActiveX server project. This option appears only for ActiveX server projects.

- **Version Info**—Opens the Version Info dialog box. You can enter version information for the executable file in this dialog box. LabWindows/CVI saves the version information in the executable as a standard Windows version resource. You can obtain the information

from the executable by using the Windows SDK `GetFileVersionInfo` and `GetFileVersionInfoSize` functions.

In the Version Info dialog box, the entries for **File Version** and **Product Version** must be in the following form:

`n`,`n`,`n`,`n`

where `n` is a number from 0 to 255.

- **LoadExternalModule Options**—The following options assist you in loading external modules.

  - **Enable LoadExternalModule**—Select this option if your project uses `LoadExternalModule`. This option is enabled by default. If your project does not use `LoadExternalModule`, disable this option to reduce the size of your executable. If you disable the option but still use `LoadExternalModule`, LabWindows/CVI prompts you to enable full support. You must rebuild your project before the changes take effect.

  - **Add Files to Executable**—Select additional module files you want to link into the **Application File**. These are modules that your project files do not directly reference but that are referenced by modules you load at run time by calling `LoadExternalModule`.

    If you force a Windows SDK import library into your project, your executable might not start up successfully. The Windows SDK import libraries contain functions that are not present on all versions of Windows. If the DLL on your system does not export all the functions in the import library, your executable will fail at startup. Instead of forcing an import library into your executable, you can force only the functions you need into the executable. To force specific functions into your executable, create a table of function pointers and add the functions to the table. For example, to force references to `CreateWindow` and `GetFreeDiskSpace`, you can add the following code to a source file in your project.

    ```
    void* ReferenceFunctionsTable[] = {
        CreateWindow,
        GetDiskFreeSpace,
    }
    ```

- **OK**—Accepts the current inputs and closes the dialog box.
- **Cancel**—Cancels the operation and closes the dialog box.

Target Settings for DLLs

When you set the **Target Type** to **Dynamic Link Library** and select **Build»Target Settings**, the Target Settings dialog box has the following options:

• **DLL File**—The name of the DLL files for the debug and release versions of your program. Changing the value of **DLL File** allows you to edit the filename for the debug and release configurations. You can use the **Browse** button to select an existing filename.

📝 **Note** If you select the **LabVIEW Real-Time Only** option for **Run-Time Support**, the filename you select must conform to the 8.3 DOS standard to be downloadable to a LabVIEW Real-Time board.

• **Import Library Base Name**—Normally, the name of the import library is the same as the name of the DLL except that the extension is .lib. There might be some cases, however, when you want to use a different name. For example, you might want to append _32 to the name of your DLL to distinguish it as a 32-bit DLL but not append it to the import library name. This is, in fact, the convention used for VXI*plug&play* and IVI instrument driver DLLs. If you want to enter a different name for the import library, disable the **Use Default** option. Enter a name without any directory names.

• **Where to copy DLL**—This option instructs LabWindows/CVI to copy the DLL to a different directory after creating it. Choose from the following items:

– Do not copy

– Windows System directory

– IVI Standard Root directory (the bin directory under the IVI framework directory)

– VXI*plug&play* directory (the bin directory under the VXI*plug&play* framework directory)

• **Run-Time Support**—The run-time support for your DLL. If you select the **Instrument Driver Only** item or the **LabVIEW Real-Time Only** item, your project does not link to the entire set of LabWindows/CVI libraries but instead links to a smaller set of functions.

DLLs you create with the **Full Run-Time Engine** item selected use the LabWindows/CVI Run-time Engine DLL, cvirte.dll. DLLs you create with the **Instrument Driver Only** item selected use instrsup.dll instead of cvirte.dll. The **Instrument Driver Only** option is particularly useful for creating instrument driver DLLs. The option allows other applications to use instrument driver DLLs without having to load the large LabWindows/CVI Run-time Engine DLL. DLLs you create with the **LabVIEW Real-Time Only** item selected use cvi_lvrt.dll instead of cvirte.dll or instrsup.dll. The **LabVIEW Real-Time Only** option is useful for creating DLLs that will be downloaded and run on the LabVIEW Real-Time hardware. If you use a stand-alone compiler and want to use instrsup.dll, include cvi70\extlib\instrsup.lib in your external compiler project instead of

`cvirt.lib` and `cvisupp.lib`. Remember that when you use an external compiler, you link to that compiler's ANSI C library.

`instrsup.dll` contains functions from the following libraries:

– Formatting and I/O Library (except `ArrayToFile` and `FileToArray`)

– RS-232 Library (except `InstallComCallback`)

– Utility Library (selected functions only)

– ANSI C Library

Your project also can link to the following libraries:

– Analysis or Advanced Analysis Library

– GPIB/GPIB 488.2 Library

– VXI Library

– VISA Library

– IVI Library

– NI-DAQmx Library

– Traditional NI-DAQ Library—If your project files call the
  `Config_Alarm_Deadband`, `Config_ATrig_Event_Message`,
  `Config_DAQ_Event_Message`, `DIG_Change_Message_Config`,
  `Get_DAQ_Event`, or `Peek_DAQ_Event` functions in the Traditional NI-DAQ
  Library, you will get link errors when you build your program in LabWindows/CVI.
  The link errors occur because the preceding functions use the LabWindows/CVI
  User Interface Library internally, which is not available in the `instrsup` and
  `cvi_lvrt` Run-time Engine DLLs. Your project files can call any of the other
  functions in the Traditional NI-DAQ Library.

If you use a stand-alone compiler and want to use any of these libraries, refer to
Chapter 3, *Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference
Manual*.

If you use the **Create Distribution Kit** command on a project that you link for
instrument driver support only, LabWindows/CVI automatically selects **Instrument
Driver Only** as the **Run-Time Engine Support**.

The following Utility Library functions are in `instrsup.dll`.

– `Beep`

– `DateStr`

– `Delay`

– `SyncWait`

– `Timer`

– `TimeStr`

- `RoundRealToNearestInteger`

- `TruncateRealNumber`

- `InStandaloneExecutable`

- `CVIRTEHasBeenDetached`

instrsup.dll does not support the Standard Input/Output window. Functions such as `FmtOut` or `ScanIn` return errors when you use them with `instrsup.dll`.

All the functions in `instrsup.dll` are multithread safe.

`cvi_lvrt.dll` contains the TCP Support Library and the libraries in `instrsup.dll` with the following restrictions:

–   The RS-232 Library is not available.

–   The file I/O and console I/O functions in the ANSI C, Formatting and I/O, and Utility Libraries are not supported and return run-time errors. Some of the functions affected by this are `printf`, `scanf`, `fprintf`, `fscanf`, `ScanIn`, `ScanFile`, `FmtOut`, `FmtFile`, `Beep`, `tmpfile`, and `tmpnam`.

–   Only the `C` locale is recognized, and all case-sensitive functions such as `stricmp` and `tolower` perform conversions only from characters a–z to characters A–Z, and vice-versa, and perform comparisons only using this set of characters. Locale specific case-sensitivity is not honored.

–   The multibyte to wide-character conversion functions `mbtowc` and `mbstowcs` are not available. The wide-character to multibyte conversion functions `wctomb` and `wcstombs` are not available.

–   The ANSI C environment functions `getenv`, `_putenv`, and `system` are not supported and return run-time errors.

–   DLLs that link to `cvi_lvrt.dll` can be built only in the Release configuration, and you cannot debug them.

You can link your **LabVIEW Real-Time Only** project to the VISA Library, the Traditional NI-DAQ Library, or the NI-CAN Library. If you do so, however, you must not combine calls to these LabWindows/CVI libraries with calls to the corresponding libraries provided by the LabVIEW Real-Time development environment.

If you use a stand-alone compiler and want to use `cvi_lvrt.dll`, include `cvi70\extlib\cvi_lvrt.lib` in your external compiler project instead of `cvirt.lib` and `cvisupp.lib`. Remember that when you use an external compiler, you link to that compiler's ANSI C library.

If you use the **Create Distribution Kit** command on a project that you link for **LabVIEW Real-Time Only Support** only, LabWindows/CVI automatically selects **LabVIEW Real-Time Only** as the **Run-Time Engine Support**.

All the functions in `cvi_lvrt.dll` are multithread safe.

- **Embed Project .UIRs**—Embeds `.uir` files into the DLL. Enabling this option allows you to ship only the DLL, instead of including the `.uir` files also. Make sure that you include all `.uir` files loaded through `LoadPanel` or `LoadPanelEx` in your project if you want to ship only one DLL. If you pass an absolute filename to `LoadPanel` or `LoadPanelEx`, the function will always look on disk for the `.uir` file. If you pass a simple filename, `LoadPanel` or `LoadPanelEx` will look first for an embedded `.uir`.

- **Generate Map File**—Creates a memory map for the executable. This map lists the address of every function and variable and the name of the function or variable. [This option is useful if your program crashes and provides an address where the failure occurred. You can view the map file to check which function or variable caused the crash.]

- **Register ActiveX Server After Build**—LabWindows/CVI registers the target each time that it builds the target. By default, this option is enabled for a new ActiveX server project. This option appears only for ActiveX server projects.

- **Version Info**—Opens the Version Info dialog box. You can enter version information for the DLL in this dialog box. LabWindows/CVI saves the version information in the DLL as a standard Windows version resource. You can obtain the information from the DLL by using the Windows SDK `GetFileVersionInfo` and `GetFileVersionInfoSize` functions.

  In the Version Info dialog box, **File Version** and **Product Version** must be in the following form:

  `n,n,n,n`

  where `n` is a number from 0 to 255.

- **Import Library Choices**—Choose whether to create a DLL import library for each of the compatible external compilers or to create only one for the current compatible compiler. Refer to Chapter 3, *Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*. You also can choose whether to create the import libraries in the VXI*plug&play* or IVI subdirectories instead of the directory of the DLL.

  If you choose to use the DLL directory and create an import library for each compiler, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI also creates the library for the current compatible compiler in the directory of the DLL. If you choose to create an import library only for the current compiler, LabWindows/CVI creates the file in the directory of the DLL.

  If you choose to use the VXI*plug&play* directories and create an import library for both compilers, LabWindows/CVI creates the files in the `VXIPNP\lib\msc` and `VXIPNP\lib\bc` subdirectories. If you choose to create an import library for the current compiler only, LabWindows/CVI creates the file in the appropriate subdirectory.

  If you use the IVI directories and create an import library for both compilers, LabWindows/CVI creates the files in the `IVI\Lib\msc` and `IVI\Lib\bc` subdirectories. If you choose to create an import library for the current compiler only, LabWindows/CVI creates the file in the appropriate subdirectory.

- **Type Library**—Choose whether to add a type library resource to your DLL. Also, you can choose to include links in the type library resource to a Windows help file. LabWindows/CVI generates the type library resource from a function panel (`.fp`) file. You must specify the name of the `.fp` file. You can generate a Windows help file from the `.fp` file by selecting the **Options»Generate Windows Help** command in the Function Tree Editor window.

  This feature is useful if you intend for your DLL to be used from Visual Basic.

- **LoadExternalModule Options**—The following options assist you in loading external modules.

  – **Enable LoadExternalModule**—Select this option if your project uses `LoadExternalModule`. This option is enabled by default. If your project does not use `LoadExternalModule`, disable this option to reduce the size of your DLL. If you disable the option but still use `LoadExternalModule`, LabWindows/CVI prompts you to enable full support. You must rebuild your project before the changes take effect.

  – **Add Files to DLL**—Select additional module files that you want to link into the DLL. These are modules that your project files do not directly reference but that are referenced by modules you load at run time by calling `LoadExternalModule`.

  If you force a Windows SDK import library into your project, your DLL might not load. The Windows SDK import libraries contain functions that are not present on all versions of Windows. If the DLL on your system does not export all the functions in the import library, your DLL will not load. Instead of forcing an import library into your DLL, you can force only the functions you need into the DLL. To force specific functions into your DLL, create a table of function pointers and add the functions to the table. For example, to force references to `CreateWindow` and `GetFreeDiskSpace`, you can add the following code to a source file in your project.

  ```
  void* ReferenceFunctionsTable[] = {
      CreateWindow,
      GetDiskFreeSpace,
  }
  ```

- **Exports**—The following options assist you in exporting symbols.

  – **Export What**—Indicates the current method for determining which symbols in the DLL to export to the users of the DLL. Use the **Change** button to change your choice.

  – **Change**—Select the method to use for determining which symbols in the DLL to export to the users of the DLL. You have the following choices:

    - **Include File Symbols**—You must name one or more include files that declare symbols defined globally in the DLL. The DLL exports the symbols you declare in the include files. You can select from a list of include files in the project.

- **Include File and Marked Symbols**—The DLL exports all symbols you define in the DLL with the `__declspec(dllexport)` or `export` qualifier and the symbols you declare in the include files.

- **Symbols Marked for Export**—The DLL exports all symbols you define in the DLL with the `__declspec (dllexport)` or `export` qualifier.

Note When you use the **Symbols Marked for Export** option or the **Include File and Marked Symbols** option and include in your project an object or library file that defines exported symbols, LabWindows/CVI cannot correctly create the import libraries for both of the compatible external compilers. This problem does not arise if you use only source code files in your DLL project.

- **OK**—Accepts the current inputs and closes the dialog box.
- **Cancel**—Cancels the operation and removes the dialog box.

For more information about creating DLLs, refer to Chapter 3, *Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

Target Settings for Static Libraries

When you set the **Target Type** to **Static Library** and select **Build»Target Settings**, the Target Settings dialog box has the following options:

- **Library File**—The name of the static library file for the release version of your program. Changing the value of **Library File** allows you to edit the filename for the release configuration. You can use the **Browse** button to select an existing filename.

- **Library Generation Choices**—Choose whether to create a static library for each of the compatible external compilers or create one for the current compatible compiler only. If you want to create a static library for both compilers, you must not include any object or library files in your project because such files are specific to a particular compiler.

  If you choose to create a static library for each compiler, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI also creates the library for the current compatible compiler in the parent directory.

- **OK**—Accepts the current inputs and closes the dialog box.
- **Cancel**—Cancels the operation and removes the dialog box.

Build»External Compiler Support

Use the **External Compiler Support** command to help you build your project in one of the two compatible external compilers.

The External Compiler Support dialog box contains the following options:

- **UIR Callbacks**—Creates an object or source file for you to link into your executable or DLL. The object or source file contains a list of the callback functions you specify in the `.uir` files in your project. When you load a panel or menu bar from the `.uir` file, the User Interface Library uses the list to link the objects in the panel or menu bar to their callback functions in your executable or DLL. If you specify callback function names in your `.uir` file(s), set **UIR Callbacks** to **Source File**, enter the name of the source file to create, and click **Create**. In the future, whenever you save modifications to any of the `.uir` files in the project, LabWindows/CVI automatically updates the source file.

  You must call the `InitCVIRTE` function at the beginning of your `main`, `WinMain`, or `DLLMain` function so that LabWindows/CVI run-time libraries can initialize the list of names from the source file. If you create a DLL and any of your callback functions are defined in but not exported by the DLL, you must call `LoadPanelEx` or `LoadMenuBarEx` (rather than `LoadPanel` or `LoadMenuBar`) from the DLL.

- **Using LoadExternalModule to Load Object and Static Library Files**—Enables the section of the dialog box that you use when creating an executable or DLL that calls the Utility Library `LoadExternalModule` function to load object or static library files.

✎ **Note** This option is not necessary if you use `LoadExternalModule` to load only DLLs that you load through DLL import libraries.

  Unlike DLLs, object and static libraries can contain unresolved external references. When you use `LoadExternalModule` to load an object or static library file, LabWindows/CVI resolves these references using symbols in your executable or DLL or in previously loaded external modules. Consequently, the names of the symbols in your executable or DLL that are necessary to resolve these references must be available to the `LoadExternalModule` function.

- **CVI Libraries**—Provides information that `LoadExternalModule` requires when your run-time modules reference symbols in any of the following LabWindows/CVI libraries:
  - User Interface Library
  - RS-232 Library
  - DDE Support Library
  - TCP Support Library
  - Formatting and I/O Library
  - Utility Library

  If you use one of these libraries, include in your external compiler project the source file displayed in this indicator. This option does not apply if you use `LoadExternalModule` to load only DLLs.

- **ANSI C Library**—Provides information that `LoadExternalModule` requires when your run-time modules reference symbols in the ANSI C Library. Include in your external compiler project the source file displayed in this indicator. This does not apply if you use `LoadExternalModule` to load only DLLs.

- **Other Symbols**—Creates an object file for you to link into your executable or DLL. Select this option if your run-time modules refer to symbols other than those covered by the previous two options. Such symbols include functions or variables that you define globally in your executable or DLL and to which your object or static library run-time modules expect to link.

  – **Header File**—Insert the name of an include file that contains complete declarations of all the symbols necessary to resolve references from run-time modules.

  – **Object File**—Enter the name of the object file to create. Click **Create** to create the file. You must include this file in your external compiler project.

The bottom of the External Compiler Support dialog box contains a list of library files to include in your external compiler project. The files are as follows:

- `cvi70\extlib\cvirt.lib`
- `cvi70\extlib\cvisupp.lib`
- `cvi70\extlib\cviwmain.lib`

Use `cviwmain.lib` only when the external compiler requires you to define `WinMain`, when you do not define it in your project, and when any of the libraries the external compiler automatically links do not define it. In general, console applications do not require `WinMain`. GUI application wizards sometimes automatically include it in the source code they generate.

Build»Create Distribution Kit

Use the **Create Distribution Kit** command to make an installer application from which you can install your executable program on a target machine. **Create Distribution Kit** automatically includes all the files necessary to run your executable program on a target computer except for DLLs for National Instruments hardware, files that you load using `LoadExternalModule`, and any ActiveX servers that are not the target of your project.

Do not include DLLs for National Instruments hardware in your distribution kit. Users can install the DLLs for their hardware from the distribution disks that they obtain from National Instruments.

If you load files using `LoadExternalModule`, you must include these files manually using the **Add/Edit Group** options in the Create Distribution Kit dialog box. Refer to Chapter 4, *Creating and Distributing Release Executables and DLLs*, of the *LabWindows/CVI Programmer Reference Manual* for more information about application distribution.

The following options are available in the Create Distribution Kit dialog box.

The **Install Location** section of the Create Distribution Kit dialog box contains the following options:

- **Parent Folder**—The default directory on the target machine where your application will install. You can select from a list of common Windows and National Instruments locations. These locations automatically resolve to the actual directories on the target machine during installation. These locations can vary based on operating system version, language, and configuration. Notice that Windows XP locations are similar to Windows 2000, Windows NT typically uses `C:\Winnt\Profiles` instead of `C:\Documents and Settings\`, and Windows 98 locations also apply to Windows Me/95.

  These locations also are valid for the **Group Destination** option. You can install your application in the following locations:

  – **Common Files Directory**—The full path of the Common Files folder for the current user. A common value for this directory is `C:\Program Files\Common Files\`.

  – **Desktop Directory**—The full path of the All Users Desktop folder, which contains items on the Windows desktop. Common values for this directory are `C:\Documents and Settings\All Users\Desktop\` for Windows 2000 and `C:\Windows\Desktop\` for Windows 98.

  – **Favorites Directory**—The full path of the Favorites folder for the current user. Common values for this directory are `C:\Documents and Settings\[Logon User]\Favorites\` for Windows 2000 and `C:\Windows\Favorites\` for Windows 98.

  – **Fonts Directory**—The full path of the system Fonts folder.

  – **IVI Standard Root Directory**—The full path of the IVI Standard Root directory. If IVI is already installed on the system, the installer will use that directory; otherwise the installer will use its default directory.

✎ **Note** If you use the **Create Distribution Kit** option to create an installer for an IVI instrument driver, your installer will not be compliant with the installation requirements defined by the IVI Foundation. To create a compliant IVI Driver installer, contact National Instruments directly, or send an email to `instrument.driver@ni.com`.

  – **NI LabVIEW Directory**—The full path of the most recently installed version of NI LabVIEW. If LabVIEW is already installed on the system, the installer will use that directory; otherwise the installer will use its default directory.

  – **NI LabWindows/CVI Directory**—The full path of the most recently installed version of NI LabWindows/CVI. If LabWindows/CVI is already installed on the system, the installer will use that directory; otherwise the installer will use its default directory.

–   **Program Files Directory**—The full path of the Program Files folder. A common value for this directory is `C:\Program Files`.

–   **Programs Menu Directory**—The full path of the All Users Program Menu folder. Common values for this directory are `C:\Documents and Settings\All Users\Start Menu\Programs\` for Windows 2000 and `C:\Windows\Start Menu\Programs\` for Windows 98. This folder is located under the Windows **Start»Programs** menu.

–   **Root Directory**—The top-level directory of the local fixed disk or partition with the most available free space. The Root Directory is not necessarily the `C:` or the Windows drive. A common value for this directory is `C:\`.

–   **Send to Directory**—The full path of the SendTo folder. Common values for this directory are `C:\Documents and Settings\[LogonUser]\SendTo\` for Windows 2000 and `C:\Windows\SendTo\` for Windows 98. This folder contains the shortcuts that appear in the SendTo menu. To access the **SendTo** menu, right-click a file in Windows Explorer.

–   **Start Menu Directory**—The full path of the All Users Start Menu folder. Common values for this directory are `C:\Documents and Settings\All Users\Start Menu\` for Windows 2000 and `C:\Windows\Start Menu\` for Windows 98. This folder contains the shortcuts that appear directly under the Windows **Start** button.

–   **Startup Directory**—The full path of the All Users Startup folder. Common values for this directory are `C:\Documents and Settings\All Users\Start Menu\Programs\Startup\` for Windows 2000 and `C:\Windows\Start Menu\Programs\Startup\` for Windows 98.

–   **Temp Directory**—The full path of the Temp folder. Common values for this directory are `C:\Documents and Settings\[LogonUser]\Local Settings\Temp\` for Windows 2000 and `C:\Temp\` for Windows 98.

–   **Template Directory**—The full path of the All Users Template folder. Common values for this directory are `C:\Documents and Settings\All Users\Templates\` for Windows 2000 and `C:\Windows\ShellNew\` for Windows 98.

–   **VXI PnP Directory**—The value of the VXI*plug&play* directory. If VXI is already installed on the system, the installer will use that directory; otherwise the installer will use its default directory. A common value is `C:\VXIpnp`.

–   **VXI PnP OS Directory**—The value of the VXI*plug&play* operating system directory. If VXI is already installed on the system, the installer will use that directory; otherwise the installer will use its default directory. A common value is `C:\VXIpnp\[OS type]`.

–   **Windows Directory**—The full path of the Windows folder. Common values for this directory are `C:\Winnt` for Windows 2000 and `C:\Windows` for Windows 98.

–   **Windows System 16-bit Directory**—The full path of the folder for 16-bit system DLLs. Common values for this directory are `C:\Winnt\System\` for Windows 2000 and `C:\Windows\System\` for Windows 98.

–   **Windows System Directory**—The full path of the System folder. Common values for this directory are `C:\Winnt\System32` for Windows 2000 and `C:\Windows\System\` for Windows 98.

–   **Windows Volume Root Directory**—The root directory of the partition containing the currently running Microsoft Windows. A common value for this directory is `C:\`. Notice that this value can be different than the **Root Directory** option because Windows may not always be installed on `C:` or because the Root Directory may not always resolve to `C:\`.

•   **Sub Folder**—The subfolder that appears in the user installation. You can specify `\.` to denote the same level as the **Parent Folder**.

The **Build Information** section of the Create Distribution Kit dialog box contains the following options:

•   **Build Location**—The path into which you want to build your distribution kit. At a minimum, you must have at least 3.5 MB of space in the build location to successfully create a basic distribution kit. Creating a distribution kit directly to a 1.44 MB floppy disk is not supported. To install your application on another machine, you must include all of the files from this directory and then launch `setup.exe`.

•   **Browse**—Browses for a build location.

•   **Installation Language**—The language that the installation program uses for text during the installation.

✎   **Note**   You might receive an error if you attempt to build a Japanese distribution kit on English Windows NT 4.0/9*x*. These operating systems do not have support for the Japanese codepage installed by default. You also might receive an error at distribution kit install time on these same English versions of Windows. While there are several ways to install Japanese codepage support on your system, the following is a possible method:

1.   Install Microsoft Internet Explorer 5.0 or later.

2.   Go to `http://windowsupdate.microsoft.com` and follow the links to the products update section.

3.   Download Japanese Language Support for Internet Explorer and follow the installation intructions.

4.   Reboot, restart LabWindows/CVI, and retry the distribution kit build.

•   **Run-Time Engine Support**—The run-time engine to include in your distribution kit. You must match this choice to the run-time engine that your project is built with in the Target Settings dialog box. Selections other than **Full Run-Time Engine** or **All Engines**

cause other options on the distribution kit to be unavailable, because those options require
Full Run-Time Engine support to function. You can select the following options:

–   **Full Run-Time Engine**—Includes the full LabWindows/CVI Run-time Engine
    DLL (`cvirte.dll`) and its supporting files. If you choose **Install in Windows
    System Directory**, the run-time engine is installed into whatever the Windows
    System directory resolves to on the target machine. If you choose **Install in
    Application Directory**, the run-time engine is installed in the same directory as your
    application.

–   **Instrument Driver Only**—Includes support for instrument drivers. If you install
    instrument driver support, your project does not link to the entire set of
    LabWindows/CVI libraries but to a smaller set of functions. When you select this
    option, your distribution will not include the LabWindows/CVI Run-time Engine
    DLL, `cvirte.dll`. Instead, your distribution includes `instrsup.dll`, which is
    much smaller. This command allows other applications to use instrument driver
    DLLs without having to load the large LabWindows/CVI Run-time Engine DLL.

    If you choose **Install in Windows System Directory**, `instrsup.dll` is installed
    into whatever the Windows System directory resolves to on the target machine.
    If you choose **Install in Application Directory**, `instrsup.dll` is installed in the
    same directory as your application.

    Also, if you create a distribution kit for a project that you link for instrument driver
    support only in the Target Settings dialog box, LabWindows/CVI automatically
    enables the option to install `instrsup.dll`. Do not include `instrsup.dll`
    directly into a file group—use this option instead.

    For more information about `instrsup.dll`, refer to the *Target Settings for DLLs*
    section.

–   **LabVIEW Real-Time Only**—If you select **LabVIEW Real-Time Only**,
    `cvi_lvrt.dll` is installed instead of `cvirte.dll` or `instrsup.dll`. Use this
    option for DLLs that will be downloaded and run on the LabVIEW Real-Time
    hardware.

    For more information about `cvi_lvrt.dll`, refer to the *Target Settings for DLLs*
    section.

–   **None**—Does not include any run-time engine support.

–   **All Engines**—Includes support for all three of the preceding run-time engines.

    Because it is not known at distribution kit build time if the target machine already
    has a run-time engine, NI recommends that you not choose **None** for your run-time
    engine support. Also, do not include run-time engine DLLs directly in a file group,
    because the DLL might not run correctly or it might not be properly installed.

•   **Run-Time Engine Install Location**—Selects the installation location for the run-time
    engine. If you choose **Install in Windows System Directory**, the run-time engine files
    are installed into the directory that the Windows `system` directory resolves to on the

target machine. If you choose **Install in Application Directory**, all run-time engine and supporting files are installed in the same directory as your application. The low-level support driver files cannot be installed in the same directory as your application.

- **Install DataSocket Support**—Set this option to **If Needed** to include the DataSocket support files in your distribution only if your project uses the DataSocket Library. Set this option to **Always** to include the DataSocket files in your distribution regardless of whether your project uses the DataSocket Library. Set this option to **Never** if you do not want to include DataSocket files in your distribution. DataSocket requires Internet Explorer 5.0 or later installed on the target machine to execute properly.

- **Install NI-Report Support**—Set this option to **If Needed** to include the NI-Report support files and ActiveX server in your distribution only if your project uses the NI-Report instrument driver. Set this option to **Always** to include the NI-Report files and ActiveX server in your distribution regardless of whether your project uses the NI-Report instrument driver. Set this option to **Never** if you do not want to include NI-Report files in your distribution.

- **Install Low-Level Support Driver**—Selects whether to install the LabWindows/CVI low-level support driver on the user's computer. The Utility Library functions shown in the following table require the LabWindows/CVI low-level driver. If you use any of these functions in your application, you must enable this option.

Function	Platforms that Require the Low-Level Support Driver
inp	Windows 2000/NT/XP
inpw	Windows 2000/NT/XP
outp	Windows 2000/NT/XP
outpw	Windows 2000/NT/XP
ReadFromPhysicalMemory	All
ReadFromPhysicalMemoryEx	All
WriteToPhysicalMemory	All
WriteToPhysicalMemoryEx	All
MapPhysicalMemory	All
UnMapPhysicalMemory	All
DisableInterrupts	Windows 98
EnableInterrupts	Windows 98
DisableTaskSwitching	Windows 98

The LabWindows/CVI Run-Time Engine loads the low-level support driver automatically at startup if the support driver is present on the machine.

• **Install ActiveX Container Support**—Installs ActiveX container support on the user's computer. If your project includes any ActiveX controls, you must enable this option. You also must enable **Register Files As ActiveX Servers** if you distribute ActiveX controls in your project.

• **Include Hardware Configuration**—Includes an NI Measurement & Automation Explorer (MAX) hardware configuration export file. Selecting this option enables the **Export** button, which you can use to generate a hardware configuration file using MAX. Additionally, this option includes the specified export file in the distribution kit for installation on the target machine. The export file is installed in the directory you specified in the options of the **Install Location** section. After you create the configuration file the first time, you do not need to run the MAX export again, unless you need to include any hardware or task changes that you have made.

At installation time on the target machine, if MAX version 3.0 or later is installed, this configuration file causes the MAX Configuration Import Wizard to launch automatically after a reboot. If MAX version 3.0 or later is not installed on the target machine, then a warning dialog appears and the configuration is not imported, but the install continues normally.

• **Export**—Launches the MAX Configuration Export Wizard. You must have MAX version 3.0 or later installed to use this option. After you complete the export wizard, the name of the export file you generated appears in the indicator to the left of this button. If you make any hardware or task changes that you want included in your distribution kit, you must run the MAX export wizard again.

The **File Groups** section of the Create Distribution Kit dialog box has the following options:

• **File Groups**—Separates the files in your distribution kit into groups. You must assign a destination directory to each group. The distribution kit creates the directories on the target machine and places each of the file groups in its assigned directory. You can set each of the options to the right of the list box to different values for each file group. File groups for the files in your project are automatically created for you. If you enable the **Embed Project .UIRs** option in the Target Settings dialog box, you do not need to add .uir files to the file groups.

• **Add Group**—Adds a new group to your distribution kit.

• **Edit Group**—Edits the selected group.

• **Delete Group**—Deletes the selected group from your distribution kit.

• **Create Shortcuts**—Creates shortcuts that contain icons for files in the selected file group. The installation program will install the embedded icons for .exe files and the default icons for .com, .pif, .lnk, .bat, .cmd, .hlp, .txt, .doc, .wri, .xls, .ppt, .htm, .html, .vsd, .xml, .pdf, and .reg files.

To create a shortcut to the same file in more that one location, you can add the file to a new file group with the same **Group Destination** and change the shortcut in that new group to be created in the additional location. Items created on the **Start** menu are shortcuts, so they do not have file extensions. If you create shortcuts for two files of the same base name, such as `test.exe` and `test.hlp`, two shortcuts with the name `test` are created, and one overwrites the other. Make sure that you have different names for each shortcut created in the same location.

You can create shortcuts in the following locations:

– **Start Programs (+Sub-Folder)**—LabWindows/CVI creates shortcuts in **Start»Programs»Sub-Folder»***Shortcut Name*. You can change the **Start Programs Sub-folder** in the Advanced Distribution Kit Options dialog box. The default for the subfolder is usually the name of the current LabWindows/CVI project.

– **Start Programs (Top-Level)**—LabWindows/CVI creates shortcuts in **Start»Programs»***Shortcut Name*.

– **Start (Top-Level)**—LabWindows/CVI creates shortcuts in the top portion of the menu directly under the **Start** button.

– **None**—No shortcuts are created for the files in this group.

– **Windows Startup**—LabWindows/CVI creates shortcuts in the Windows Startup folder. The items in the Windows Startup folder execute automatically every time a user logs into the system.

– **Desktop**—LabWindows/CVI creates shortcuts on the user's Windows desktop.

– **Send to Menu**—LabWindows/CVI creates shortcuts in the user's **Send To** menu. The **Send To** menu appears when you right-click a file in Windows Explorer.

• **Group Destination**—Sets the root destination directory for the selected group. You can select the destination directory to be either the application install location or common Windows locations. These locations automatically resolve to the actual directories on the target machine during the install. For a complete description of the directories you can choose from, refer to the **Parent Folder** option.

– **Application Directory**—The directory in which your application will be installed.

• **Relative Path**—Assigns a relative path based on the group destination directory in which to install the selected file group.

Note   Using the `..\` syntax can cause unexpected results if you attempt to reference a directory that is "higher" than one of the common Windows locations. For example, if you set the **Group Destination** to **Windows System Directory** and the **Relative Path** to `..\Test`, you expect the Group Destination to resolve to `C:\Windows\System` and the relative path to resolve to `C:\Windows\Test`. However, due to the behavior of the installer, the relative path resolves to `C:\Test` because the common Windows locations behave as if they were one single directory, no matter how many elements are in the path.

Set the **Group Destination** to **Windows Directory**, and set the **Relative Path** to Test to obtain the expected location.

• **Distribute Objects/Libraries for Both Compilers**—Helps distribute object files, static libraries, and DLL import libraries for both compatible external compilers. When enabled, this option affects all the .obj and .lib files listed in the selected file group. LabWindows/CVI includes two versions of each file in the distribution kit. LabWindows/CVI expects these versions to be in subdirectories under the specified location of each file. The subdirectories must be named msvc or borland. For example, if you specify the file c:\myapp\distr\big.lib in a file group and you enable the **Distribute Objects/Libraries for Both Compilers** option, when LabWindows/CVI creates the distribution kit, you must have the following files on your disk:

   – c:\myapp\distr\msvc\big.lib
   – c:\myapp\distr\borland\big.lib

   To have LabWindows/CVI create these files automatically, enable the corresponding option in the Target Settings dialog box. The installation program prompts the user to choose one of the compatible external compilers. The installation program installs only the files for the compiler the user chooses at distribution kit install time.

   You might want to use this feature if you distribute modules for use with the LabWindows/CVI development environment or external compilers. If you distribute a turnkey application, this feature is not necessary.

Note Do not use this feature for distributing DLL import libraries for VXI*plug&play* instrument drivers. When installing a VXI*plug&play* instrument driver, you must install two import libraries: one compatible with Visual C/C++ and the other compatible with Borland C/C++. The two import libraries must be installed in the VXIPNP\lib\msc and VXIPNP\lib\bc subdirectories. LabWindows/CVI sets this up automatically for you if you enable the **Options»IVI/VXIplug&play Style** command and select **Options»Create DLL Project** in the Function Tree Editor. For more information about distributing VXI*plug&play* instrument drivers, refer to the *Creating a Distribution Kit for VXIplug&play Instrument Drivers* section.

Note If you use the **Create Distribution Kit** command to distribute DLL import libraries for IVI instrument drivers, do not use the **Distribute Objects/Libraries for Both Compilers** option. To distribute IVI instrument drivers, you must install two import libraries: one compatible with Visual C/C++ and the other compatible with Borland C/C++. The two import libraries must be installed in the IVI\Lib\msc and IVI\Lib\bc subdirectories. LabWindows/CVI sets this up automatically for you if you enable the **Options»IVI/VXIplug&play Style** command and select **Options»Create DLL Project** in the Function Tree Editor for a .fp file in the IVI framework directory. For more information about distributing IVI instrument drivers, refer to the *Creating a Distribution*

Kit for IVI Instrument Drivers section and the *LabWindows/CVI Instrument Driver Developers Guide*.

- **Register Files As ActiveX Servers**—Enable this option if you distribute any ActiveX servers, such as ActiveX controls, in your project. If you use an ActiveX server in your project, you must explicitly include the control in the **File Groups** list box and enable the **Register Files As ActiveX Servers** option. Enabling **Register Files As ActiveX Servers** causes LabWindows/CVI to attempt to register all the files in the file group, regardless of file type, as ActiveX servers. The installation does not show any warnings or errors for files that failed to successfully register as ActiveX servers.

  If your project itself is an ActiveX server, the distribution kit registers the target as an ActiveX server, regardless of whether you enable this option.

The Create Distribution Kit dialog box also includes the following options:

- **Advanced**—Opens the Advanced Distribution Kit Options dialog box.
- **Default**—Resets all options in the Create Distribution Kit dialog box to their default values. When you create a new distribution kit, you can click **Default** to undo changes you have made to the options in the dialog box.

📝 **Note** When you use the Create Distribution Kit dialog box to modify an existing distribution kit, **Default** replaces your existing file groupings and settings with default values. If you click **Default** in error, click **Cancel** in the dialog box that appears to prevent this change to your distribution kit.

- **Build**—Builds your distribution kit.
- **Cancel**—Cancels the Create Distribution Kit operation.

Refer to the *Customizing Create Distribution Kit Installers in LabWindows/CVI* document for information about how you can customize your distribution kit.

Creating a Distribution Kit for VXI*plug&play* Instrument Drivers

To create a distribution kit that properly installs your VXI*plug&play* instrument driver, you can use the **Options»Create DLL Project** command in the Function Tree Editor to automatically generate a LabWindows/CVI project for your instrument driver DLL. If you enable the **Options»IVI/VXI plug&play Style** command in the Function Tree Editor when you create your project, the project created will contain all the necessary Distribution Kit settings. You must add your Soft Front Panel executable and any associated files to the file group, called Soft Front Panel Files, and specify the executable to run after setup in the Distribution Kit Advanced Options dialog box.

Default Installation Location

Set the **Parent Folder** to **VXI PnP OS Directory**, and specify the instrument name, such as tktds3xx, as the **Sub Folder** in your distribution kit.

Soft Front Panel

You always must specify the Soft Front Panel as the first group in your distribution kit. In addition, always specify your Soft Front Panel as the **Executable to Run After Setup** in the Advanced Distribution Kit Options dialog box. Install the Soft Front Panel file group to the application directory and place any extra files needed by the Soft Front Panel (.uir files, help files for the Soft Front Panel, and so on) in the first file group with the Soft Front Panel executable.

File Groups

Verify that the following file groups exist in your distribution kit. What you name the groups is not important, although the following names are good suggestions. For the following examples, assume that instr is the name of the instrument.

• Soft Front Panel Files

   – This group should contain all the files necessary to run your soft front panel (executable, .uir files, and so on).

   – Install to application directory (no relative path).

• Instrument Driver Files

   – This group should contain the following files:

      • instrument source (instr.c)

      • instrument documentation (instr.doc)

      • instrument function panels (instr.fp)

      • instrument help (instr.hlp)

   – Install to application directory (no relative path).

• Instrument Include Files

   – This group should contain only the instrument header file (instr.h).

   – Install to application directory.

   – Set the relative path to ..\include.

• Instrument KnowledgeBase

   – This group should contain the knowledge base file for the instrument (instr.kb).

   – Set the group destination to **VXI PnP Directory**.

   – Set the relative path to **kbase**.

- Instrument Driver DLL
  - This group should contain the instrument DLL.
  - Install to application directory.
  - Set the relative path to `..\bin`.
- Import Library (Borland)
  - This group should contain the Borland C++ import library for your DLL (`instr.lib`).
  - Install to application directory.
  - Set the relative path to `..\lib\bc`.
- Import Library (Microsoft)
  - This group should contain the Microsoft Visual C++ import library for your DLL (`instr.lib`).
  - Install to application directory.
  - Set the relative path to `..\lib\msc`.

Creating a Distribution Kit for IVI Instrument Drivers

Note   If you use the **Create Distribution Kit** option to create an installer for an IVI instrument driver, your installer will not be compliant with the installation requirements defined by the IVI Foundation. To create a compliant IVI Driver installer, contact National Instruments directly, or send an email to `instrument.driver@ni.com`.

To create a distribution kit that installs your IVI instrument driver, you can use the **Options»Create DLL Project** command for `.fp` files in the IVI framework directory to automatically generate a LabWindows/CVI project for your instrument driver DLL. If you enable the **Options»IVI/VXIplug&play Style** command when you create your project, the project created will contain all the necessary Distribution Kit settings. You must add your Soft Front Panel executable and any associated files to the file group, called Soft Front Panel Files, and specify the executable to run after setup in the Distribution Kit Advanced Options dialog box.

Default Installation Location

Set the **Parent Folder** to **IVI Standard Root Directory** and specify the instrument name, such as `tktds3xx`, as the **Sub Folder** in your distribution kit.

Soft Front Panel

You always must specify the Soft Front Panel as the first group in your distribution kit. In addition, always specify your Soft Front Panel as the **Executable to Run After Setup** in the Advanced Distribution Kit Options dialog box. Install the Soft Front Panel file group to the application directory, and place any extra files needed by the Soft Front Panel (`.uir` files, help files for the Soft Front Panel, and so on) in the first file group with the Soft Front Panel executable.

File Groups

Verify that the following file groups exist in your distribution kit. What you name the groups is not important, although the following names are good suggestions. For the following examples, assume that `instr` is the name of the instrument.

- Soft Front Panel Files
  - This group should contain all the files necessary to run your soft front panel (executable, `.uir` files, and so on).
  - Install to application directory (no relative path).
- Instrument Driver Files
  - This group should contain the following files:
    - instrument source (`instr.c`)
    - instrument function panels (`instr.fp`)
  - Install to application directory (no relative path).
- Instrument Include Files
  - This group should contain only the instrument header file (`instr.h`).
  - Install to application directory.
  - Set the relative path to `..\..\include`.
- Instrument Driver DLL
  - This group should contain the instrument DLL.
  - Install to application directory.
  - Set the relative path to `..\..\bin`.
- Import Library (Borland)
  - This group should contain the Borland C++ import library for your DLL (`instr.lib`).
  - Install to application directory.
  - Set the relative path to `..\..\lib\bc`.

- Import Library (Microsoft)
    - This group should contain the Microsoft Visual C++ import library for your DLL (`instr.lib`).
    - Install to application directory.
    - Set the relative path to `..\..\lsib\msc`.

Advanced Distribution Kit Options

The Advanced Distribution Kit Options dialog box has the following options:

- **Executable Filename**—The name of an executable file to run after the user installation is complete. Use the **Select** button to select a file that you have already added to one of the file groups.

    If you want to run an application that already exists on the target system, create and include, in one of your file groups, a `.bat` batch file that specifies the application you want to run. You can use the Windows START utility in the batch file before the application name if you do not want the installer to wait on the launched application to exit. Choose this batch file as the executable file to run after installation.

- **Command Line Arguments**—The command line arguments to pass to the executable to run after the installation is complete. Use the this option to specify parameters to pass to the program to run after setup. The following special variables are available:

    - `%dest`—The application installation directory chosen by the user.
    - `%src`—The directory that contains `setup.exe`.
    - `%group`—The installation program group name.
    - `%name`—The installation name.
    - `%compiler`—The compiler chosen by the user if you enable the **Distribute Objects and Libraries For All Compilers** option. The compiler will be `borland` or `msvc`.
    - `%%`—Use this variable when you need to place a single `%` in your command line.

    At installation time, these variables will be replaced with the proper values before the arguments are passed to the executable.

📝 **Note** Put quotes around items that might contain spaces to ensure that your program receives the arguments correctly. The variables that can contain spaces are `%group`, `%name`, `%src`, and `%dest`. For example, assume the following command line arguments:

```
-I "%dest\myapp.txt" /from "%src" "%group" "%name"
```

The actual command line passed to the executable to run after setup is structured as follows (values vary based on user choices and installation title options):

```
-I "c:\myapp\myapp.txt" /from "c:\temp" "My App" "My App Setup"
```

- **Start Programs Sub-Folder**—The name of the shortcut subfolder created in **Start»Programs** during the installation. This option is available only if you select **Start Programs (+Sub-Folder)** from the **Create Shortcut** option in the Create Distribution Kit dialog box. If you select the **Use Default** option, LabWindows/CVI uses the following priority to determine the program group name.

  If the project target is an executable and you entered an application title in the Target Settings dialog box, LabWindows/CVI uses the application title.

  Otherwise, if you created the target executable, DLL, or static library, LabWindows/CVI uses the base filename of the target.

  Otherwise, LabWindows/CVI uses the base name of the project file.

- **Installation Name**—The installation window title and the text displayed in the upper part of the installation window. If you select the **Use Default** option, LabWindows/CVI sets the name using the same priority as for the **Start Programs Sub-Folder**.

- **Installer GUID**—The Globally Unique Identifier (GUID) used to identify this installer to the Windows Installer (MSI) engine on the target computer. Changing the GUID allows another version of the distribution kit to be installed on the same computer at the same time. Keep the following points in mind when you change the GUID of the distribution kit.

  – A GUID change does not cause an already installed version of your distribution kit to be removed automatically during the installation on the target machine. Instead, changing the GUID allows your project to be installed side-by-side with a previously installed version of your program. For example, after completing the 1.0 version of your project, you can generate a new GUID, which would allow the 2.0 version of your project to be installed on the target computer at the same time.

  – When you change the GUID, the installer adds an additional entry in the Add/Remove Programs dialog box in the Windows Control Panel. If you change the GUID, National Instruments recommends that you also change the **Installation Name** and **Start Program Sub-Folder** to help distinguish one version from another.

  – NI recommends that you also change the default **Install Location** option in the Create Distribution Kit dialog box. Doing so minimizes the chance that two different distribution kits will install the same file(s) to the same directory.

- **Generate New GUID**—Generates a new GUID for the current distribution kit installer. NI recommends that you use this option to generate the new GUID instead of editing the existing GUID. Using this option guarantees a unique GUID.

- **Install 3D Graph Control**—Set this option to **If Needed** to include the NI 3D Graph ActiveX control in your distribution only if your project uses the 3D Graph function panel. Set this option to **Always** to include NI 3D Graph ActiveX control in your distribution, regardless of whether your project uses the 3D graph. Set this option to **Never** if you do not want to include the NI 3D Graph ActiveX control under any condition.

Run Menu for the Workspace Window

This section explains how to use the commands in the Workspace window **Run** menu. You can use commands in the **Run** menu to run your program, step through code, and assign breakpoints.

Run»Debug

If you select **Build»Target Type»Executable**, the **Debug** menu item runs the project's target executable for the currently selected configuration. Set the active configuration using the **Build»Configuration** option. If you select **Build»Target Type»Dynamic Link Library**, the **Debug** menu item runs the executable you specify with the **Run»Specify External Process** menu item. Before running the executable, the **Debug** menu item compiles any source files that must be compiled and builds the project's target executable or DLL if you made changes since you last built the target DLL or executable.

Run»Continue

Use the **Continue** command to resume program execution in a breakpoint state.

Run»Step Over

Use the **Step Over** command to execute an outlined statement in a breakpoint state. If the program last suspended on a function call statement, **Step Over** executes the entire function and then enters a breakpoint state on the statement following the function call. If LabWindows/CVI encounters a breakpoint within the function call, **Step Over** pauses at the breakpoint.

Run»Step Into

The **Step Into** command is similar to the **Step Over** command except that after the program suspends operation at a function call, **Step Into** enters the function and suspends at the first statement of the function. **Step Into** can enter a function only if you define it in a source file. Otherwise, **Step Into** executes the entire function and suspends execution on the statement following the function call.

Run»Finish Function

The **Finish Function** command resumes execution through the end of the current function and breakpoints on the next statement.

Run»Terminate Execution

Use the **Terminate Execution** command to terminate a program that is in a breakpoint state.

Run»Break On

Use this command to view a submenu in which you can set additional break conditions for programs:

- **First Statement**—This option specifies that LabWindows/CVI enter a breakpoint state on the first executable statement in your source code.

- **Library Errors**—This option specifies that a breakpoint occurs when any function call to a LabWindows/CVI library results in an error.

- **First Chance Exceptions**—This option determines what LabWindows/CVI does when your program causes an exception. If you enable this option, LabWindows/CVI suspends your program before it can catch and handle the exception. In general, disable this option if you are debugging code that generates exceptions that your code catches and handles. If LabWindows/CVI suspends your program before your program catches the exception, you can use the **Run»Continue** command to resume execution of your program.

Run»Breakpoints

The **Breakpoints** command opens the Breakpoints dialog box, which contains a list of the breakpoints in the workspace. Also, you can open this dialog box by right-clicking the line icons column in a Source window and selecting **Breakpoints** from the context menu. The Breakpoints dialog box contains the following options:

- **Add/Edit Item**—Edits a single breakpoint with the Edit Breakpoint dialog box.

- **Go to Line**—Highlights the source code location of the currently selected breakpoint.

- **Delete Item**—Deletes the currently selected breakpoint.

- **Delete All**—Deletes all the breakpoints.

- **Disable All**—Forces LabWindows/CVI to ignore all the breakpoints. The breakpoint icons in the Source window change color to indicate that you disabled them.

- **Enable All**—Activates all the breakpoints. The breakpoint icons in the Source window change color to indicate that you enabled them.

- **OK**—Accepts the current breakpoint attributes and closes the dialog box.

- **Cancel**—Cancels the current operation and closes the dialog box.

For more information about breakpoints, refer to the *Introduction to Breakpoints and Watch Expressions* section of Chapter 4, *Source and Interactive Execution Windows*.

Run»Stack Trace

You can use the **Stack Trace** command only during a breakpoint state. **Stack Trace** opens a dialog box that lists the currently active functions in the program, displaying the most recently called function at the top and the initial function at the bottom. If you highlight a function in

the list and select **Display**, a Source window appears with the file that contains that function. LabWindows/CVI highlights the last statement that your program executed in that function.

Run»Up Call Stack

You can use the **Up Call Stack** command only during a breakpoint state. **Up Call Stack** moves up one level in the function call stack.

Run»Down Call Stack

You can use the **Down Call Stack** command only during a breakpoint state. **Down Call Stack** moves down one level in the function call stack.

Run»Specify External Process

This command applies only when you select **Build»Target Type»Dynamic Link Library**. Use the **Specify External Process** command to specify a stand-alone executable that uses your DLL. When you execute the command, a dialog box appears in which you enter the pathname and command line arguments to an external program. The **Run Project** item in the **Run** menu then changes to **Debug *xxx*.exe**, where *xxx*.exe is the filename of the external program. When you execute the **Debug *xxx*.exe** command, LabWindows/CVI starts the external process and attaches to it for debugging. If you have set any breakpoints in the source files for the DLL, LabWindows/CVI honors them.

LabWindows/CVI stores external program pathname and command line arguments in the workspace.

Run»Execute

The **Execute** command launches the executable for the active configuration without attaching the debugger to the executable. You must create the executable, using the **Build»Create** menu item, before you use this command. This command is dimmed if the **Target Type** for the project is **Dynamic Link Library** or **Static Library**.

Run»Command Line

Use the **Command Line** command to enter the command line arguments for your program. When you run your program in the LabWindows/CVI environment, LabWindows/CVI passes the command line arguments to your `main` function in the **argc** and **argv** parameters.

If your project makes a DLL, you can pass command line arguments to an external program that you run to debug the DLL. Specify the external program pathname and command line arguments using **Run»Specify External Process**.

Run»Threads

The **Threads** command opens a dialog box listing the threads in the program you are debugging. Use this dialog box to select the threads whose local variables and call stack you want to view. When you select a thread from this dialog box and click **OK** to close the dialog box, LabWindows/CVI displays the local variables for the selected thread in the variable display and displays the current source position of the thread in a Source window. The **Stack Trace**, **Up Call Stack**, and **Down Call Stack** commands display information about the currently selected thread. The watch display shows the thread-specific values of the expressions in the Watch window.

Run»Loaded Modules

Use this command to open the Loaded Modules dialog box, which displays the loaded executables, DLLs, and drivers used by the process that you are debugging. The address range and path of the executable, DLL, or driver also are shown.

Instrument Menu for the Workspace Window

The **Instrument** menu is a dynamic menu. This menu contains a list of the loaded instrument drivers and commands to load, unload, and edit instruments. When you load an instrument, its name appears in the list. When you unload an instrument, its name disappears from the list. When you select an instrument name in the **Instrument** menu, you can access its function panels. Instrument drivers listed in this menu also appear in the Instruments folder under the Library Tree.

Instrument»Load

When you select the **Load** command, the Load Instrument dialog box appears. In the dialog box, the filename `*.fp` appears in **File name**. Always load instruments through the `.fp` filename. You cannot load an instrument driver unless a `.fp` file exists for it.

When you specify a `.fp` file to load, LabWindows/CVI also looks in the same directory for a program file with the same base filename. If LabWindows/CVI finds a program file, it loads the instrument driver program along with the function panels.

For IVI and VXI*plug&play* instrument drivers, the program file can be in a different directory.

File Format Conversion

If you load a `.fp` created with LabWindows for DOS, a message appears indicating that LabWindows/CVI is converting the `.fp` file to the current format. You can use the dialog box that appears after the conversion to save the converted `.fp` file to disk.

Instrument»Unload

Select the **Unload** command to the Unload Instrument dialog box, which contains a scrollable list of all the loaded instruments. From this dialog box, you can select one or more instrument drivers to unload.

Instrument»Edit

You can use the **Edit** command to edit an instrument driver program in a Source window or edit an instrument driver function tree in the Function Tree Editor. When you select the **Edit** command, the Edit Instrument dialog box appears.

The dialog box displays instrument drivers you loaded as part of the project or through the **Instrument** menu. The Edit Instrument dialog box contains the following commands:

- **Show Info**—Displays the names of the current function panel file and the attached program file. This command also shows whether these files are in the current project and if the program file is compiled. The attached program file contains the functions that are called when users operate the function panel.

- **Attach and Edit Source**—If a `.c` file with the same base name as the selected `.fp` file exists in the same directory as the `.fp` file, LabWindows/CVI loads, compiles, and attaches the `.c` file as the instrument driver program file. The `.c` file appears in a new Source window. If the file is not found, LabWindows/CVI prompts you to create a new source file.

- **Detach Program**—LabWindows/CVI detaches the instrument driver program file from the `.fp` file.

- **Reattach Program**—LabWindows/CVI detaches the current instrument driver program file, if any, from the `.fp` file. LabWindows/CVI then reloads a program file using certain rules. Refer to the *Precedence Rules for Loading the Instrument Driver Program File* section of Chapter 5, *Using Instrument Drivers*.

- **Edit Function Tree**—LabWindows/CVI displays the function tree for the selected `.fp` file.

- **Done**—Closes the Edit Instrument dialog box.

Instrument»Search Directories

Use the **Search Directories** command to list the directories that LabWindows/CVI searches when loading instruments that reference other instruments. The `.fp` files of the dependent drivers store the names of the `.fp` files of the drivers on which they depend. When loading an instrument `.fp` file that references other instrument `.fp` files, LabWindows/CVI tries to find

the referenced instrument `.fp` files and load them. LabWindows/CVI searches for each `.fp` file in the following directories and in the following order:

1. The directory of the referencing `.fp` file

2. The directories listed in the Instrument Directories dialog box

3. The subdirectories under the `cvi70\toolslib` directory

4. The `cvi70\instr` directory

LabWindows/CVI saves the Instrument Directories list from one session to another.

To find out more about referencing one instrument from another, refer to Chapter 7, *Function Tree Editor*.

You also use the Instrument Directories list when you load a project file that you have moved since you last saved it. If a `.fp` file listed in the project cannot be found using either its original pathname in the project or its location relative to the project, LabWindows/CVI searches the Instrument Directories list for a `.fp` file with the same base name.

Accessing Function Panels from the Instrument Menu

When you select an instrument name in the **Instrument** menu, the Select Function Panel dialog box appears.

The Select Function Panel dialog box shows the function panels available in the driver you selected. Class names appear in the dialog box followed by ellipses (. . .). The ellipses indicate that more functions or classes of functions exist below that class name. If you select **Flatten**, the list box shows all function panels in the instrument driver.

If you select the **Function Names** option, the list box shows the function names associated with each function panel. While in this mode, the **Alphabetize** option redisplays the function list in alphabetical order.

If you select **New Window**, the next selected function panel appears in a new window. Otherwise, LabWindows/CVI overwrites the current Function Panel window. If you disable the **Options»Open Function Panels in New Window** command in a Function Panel window, LabWindows/CVI ignores the **New Window** option.

Use **Select** to select a class name to view the functions within a class. A class can contain other classes and functions. An instrument driver can contain up to four levels of classes and functions. Each time you select a class name, the function list updates. Click the **Up** button to return to the previous level.

When you select a function from a dialog box, that function panel appears. Refer to Chapter 6, *Using Function Panels*, for more information about function panels.

To close the dialog box without opening a function panel, select **Cancel**.

The Select Function Panel dialog box contains a **Help** button. Click the button to get help information about the functions and classes listed in the dialog box. Select **Help** or press <F1> to display the Help dialog box for a selected function panel.

To close the Help dialog box, click the **Done** button, press the <Enter> key, or press the <Esc> key.

Library Menu for the Workspace Window

This section explains how to use the commands in the **Library** menu. Use the **Library** menu commands to access function panels for the LabWindows/CVI libraries. Use library function panels to interactively run library functions and insert these function calls into any open Source window. For descriptions of each LabWindows/CVI library, refer to the *LabWindows/CVI Help*.

User Libraries

You can install your own libraries into the **Library** menu. A user library has the same form as an instrument driver. Anything that can be loaded into the **Instrument** menu can be loaded as a user library, provided the program is in compiled form.

The main difference between modules you load as instrument drivers and those you load as user libraries is that you can unload instrument drivers using the **Instrument»Unload** command, but you cannot unload user libraries. Also, because user libraries must be in compiled form, you cannot edit them when they are in the **Library** menu. Refer to the *LabWindows/CVI Instrument Driver Developers Guide* for more information about writing an instrument driver.

To install user libraries, select **Library»Customize** in the Workspace window. Once a library is installed, the next time you launch LabWindows/CVI, the libraries load automatically and appear at the bottom of the **Library** menu.

Dummy .fp Files for Support Libraries

If you develop a library module to provide support functions for the modules in your project, you can install it as a user library. By doing so, you ensure that the library is always available in the LabWindows/CVI development environment. If you do not want to develop function panels for the library, create a `.fp` file without any classes or functions. In that case, LabWindows/CVI loads the library at startup but does not include the library name in the **Library** menu.

System Libraries

LabWindows/CVI includes some low-level system functions. The prototypes for the functions are in `lowlvlio.h` and the function panels are in the ANSI C Library.

You can call Windows SDK functions. If you have installed the LabWindows/CVI Full Development System from CD-ROM, you have access to the full set of Windows SDK functions. Otherwise, you have access only to a subset.

Library»Customize

Use the **Customize** command to specify optional National Instruments libraries to load automatically when you start LabWindows/CVI. You also can specify which of the optional NI user libraries to load on startup. The **Customize** command opens the Customize Library Menu dialog box, which contains National Instruments Libraries and user libraries.

National Instruments Libraries

There are five optional National Instruments libraries.

- Advanced Analysis (or Analysiss)

- Traditional NI-DAQ

- NI-DAQmx

- VXI

- GPIB/GPIB 488.2

In the **National Instruments Libraries** section of the Customize Library Menu dialog box, select the library name(s) that you want LabWindows/CVI to load into memory at startup. Changes do not take effect until the next time you launch LabWindows/CVI.

If you do not load a library, you cannot call any of the functions in that library and you cannot access any of its function panels.

If LabWindows/CVI fails to load a requested library, it is probably because LabWindows/CVI cannot find the appropriate files. Make sure that the files shown in the following table appear in the `bin` directory of the LabWindows/CVI installation directory.

Library	File
Analysis or Advanced Analysis	`analysis.lib`
Traditional NI-DAQ	`dataacq.lib`
NI-DAQmx	`nidaqmx.lib`

Library	File
VXI	`nivxi.lib`
GPIB/GPIB 488.2	`gpib.lib`

You also must install the proper hardware drivers from the National Instruments Driver CD to use the optional libraries.

Tools Menu for the Workspace Window

This section describes how to use the commands in the Workspace window **Tools** menu.

Tools»Create ActiveX Controller

Use the **Create ActiveX Controller** command to generate a new instrument driver for an ActiveX server. When you select the **Create ActiveX Controller** command, LabWindows/CVI displays the ActiveX Controller Wizard.

The ActiveX Controller Wizard contains help that describes each step of building an ActiveX controller.

Tools»Create ActiveX Server

You can use the LabWindows/CVI Create ActiveX Server Wizard to provide settings for your ActiveX Server project.

The Create ActiveX Server Wizard contains help that describes each step of building an ActiveX Server.

Tools»Edit ActiveX Server

Use the Edit ActiveX Server dialog box to create and modify objects and interfaces in your ActiveX server. You also can modify the server settings.

The Edit ActiveX Server Wizard contains help that describes each step of editing an ActiveX Server.

For more information, refer to the *Building ActiveX Servers in LabWindows/CVI* document, which describes the LabWindows/CVI ActiveX server tools and functions in more detail and contains sample code fragments and hints for server development.

Tools»Create IVI Instrument Driver

Select **Tools»Create IVI Instrument Driver** to open the Instrument Driver Development Wizard, which you can use to create the source file, include file, and function panel file for controlling an instrument. You can base the new instrument driver on one of the following items:

- An existing driver for a similar instrument

- The core IVI driver template

- An IVI instrument class template

The Instrument Driver Development Wizard copies the template or existing driver files and replaces all instances of the original instrument prefix with the prefix you select for your new driver.

Refer to the *LabWindows/CVI Instrument Driver Developers Guide* for more information about the Instrument Driver Development Wizard.

The steps you take to create a driver depend on the I/O interface you select.

The Instrument Driver Development Wizard contains help that describes each step for creating an IVI instrument driver.

Tools»Create Instrument I/O Task

Use this command to launch the NI Instrument I/O Assistant. You can use the Instrument I/O Assistant to generate code to communicate with devices such as serial, Ethernet, and GPIB instruments without using an instrument driver. The Instrument I/O Assistant generates the following three files:

- `.c` source file that contains the Run Task function and associated source code

- `.h` include file that contains the Run Task function declaration

- `.mxb` binary file that contains a description of the task that is used when you modify the task in the Instrument I/O Assistant

These generated files are added to your project.

Using the Instrument I/O Assistant

The Instrument I/O Assistant organizes instrument communication into ordered steps. You must arrange the following steps into a sequence.

- **Select Instrument**—Specify the instrument with which the task will communicate. This action must be the first step in all sequences.

- **Query and Parse**—Send a command to the instrument, read a response from the instrument, and parse the returned data.

- **Write**—Write data or commands to the instrument.

- **Read and Parse**—Read a response from the instrument and parse the returned data.

As you specify and configure each step, click the **Run this step** button to execute the operations you configure, or click the **Run** button to perform the entire sequence.

For more information about configuring the steps, click **Show Help** to view the embedded help within the Instrument I/O Assistant.

You also must specify the name of the task, the name of the Run Task function to generate, and a target directory.

When you have created a sequence and configured each step, click **OK** in the Instrument I/O Assistant. LabWindows/CVI generates three files and adds the files to your project.

Once you create an Instrument I/O Assistant task, you can use it in your project.

Editing the Instrument I/O Task

You can double-click the `.mxb` file in the Project Tree to open the Instrument I/O Assistant and edit the task. If you make changes directly to the generated files, those changes are lost when you edit the task and regenerate the files.

Using an Instrument I/O Assistant Task

To use your newly created task in a program, call the Run Task function declared in the header file as shown in the following code:

```
#include "MyIOTask.h"
double *data = 0;
RunMyIOTask (&data);
/* Analyze, log, or display your data. */
free (data);
```

Tools»Create/Edit DAQmx Tasks

This command opens the Create/Edit DAQ Tasks dialog box. Use this dialog box to start creating a new DAQmx task or to launch the NI DAQ Assistant to edit an existing task.

Creating a DAQmx Task

For new tasks, you must select whether to store the task in the project or in MAX.

Tasks stored in MAX are global to the machine on which they are created and can be used by other programs on the same machine. Tasks stored in the project are local to that project. Project-based tasks can be more convenient if you need to share a task definition among multiple developers or store the task definition in a source code control system.

When you create a new task, you must specify the measurement type for the task and the channels to add to the task. Once you select these options, the DAQ Assistant appears. The DAQ Assistant contains help that describes each step of creating and editing your task. To view the help, click **Show Help**.

Project-Based Tasks

If you select a project-based task, the DAQ Assistant generates source code to create the specified task programmatically. In addition to defining your task in the DAQ Assistant, you must specify a task name, a task creation function name, and a target directory.

When you exit the DAQ Assistant, LabWindows/CVI generates source (`.c`), header (`.h`), and binary (`.mxb`) files and adds them to the project. The generated source and header files define the task creation function. This function contains the code necessary to create and configure the specified task. The generated binary (`.mxb`) file contains a binary description of the task that is used when you edit the task in the DAQ Assistant.

Once you create a project-based task, you can use it in your program.

MAX-Based Tasks

If you store the task in MAX, you can edit it from LabWindows/CVI or you can edit it directly in MAX. You can access the task in MAX by expanding **My System»Data Neighborhood» NI-DAQmx Tasks**.

You also can create a MAX-based task by selecting **New Task** from the **Task Name** control in the DAQmx Load Task function panel.

Note   You must use MAX to create global channels.

Once you create a MAX-based task, you can use it in your program.

Editing a Task

To edit an existing task, enable the **Edit Existing Task** option and choose a MAX-based task or project-based task from the list in this dialog box. Selecting this option opens the DAQ Assistant to edit the task.

You also can edit existing tasks through the **Edit DAQ Task** context menu item in a source window. You can edit project-based tasks by right-clicking the binary (`.mxb`) file in the Project Tree and selecting **Open** or **Edit**.

Using Project-Based Tasks

To use your newly created task in a program, call the task creation function declared in the generated header file as shown in the following code:

```
#include "MyProjectBasedTask.h"
TaskHandle myTask = 0;
CreateMyProjectBasedTask (&myTask);
/* Use your DAQmx task */
DAQmxClearTask (myTask);
```

The Source window provides context menu items for working with project-based DAQmx tasks. You can access these context menu items by right-clicking a call to a task creation function in a Source window. For more information refer to the *Context Menus in Source Windows* section of Chapter 4, *Source and Interactive Execution Windows*.

Tools»Source Code Control

The **Source Code Control** submenu contains menu items that you can use to perform operations with your source code control system. LabWindows/CVI does not provide a source code control system. If you have a source code control system that implements the Microsoft standard Source Code Control Interface, you can attach a LabWindows/CVI project to your source code system using the Source Code Control Options dialog box, accessible through **Options»Environment** or **Edit»Project** for project-specific source code control options.

The exact behavior of the commands in the **Source Code Control** submenu depends on the implementation of the Source Code Control Interface that your source code control system provides and on the settings in the Source Code Control Options dialog box.

Some of the source code control commands open a Select Files dialog box when you select the commands from the Workspace window. Use the Select Files dialog box to select the files you want to include in the operation and set any available options for the operation. Click the **Advanced** button to set any options that your Source Code Control Interface provides for the operation.

Source Code Control commands and options are dimmed if the Source Code Control Interface that the source code control system provides does not support the command or option. Many of the commands transfer files between your hard disk and the Source Code Control program attached to the current LabWindows/CVI program. Depending on the command you select and the window that is active, the Source Code Control commands apply to either the list of files you select or the file that is in the active window.

- **Get Latest Version**—Get the latest version of files from the source code control project.
- **Get Latest Versions of All**—Get the latest version of all files.
- **Check Out**—Check out files from the source code control project.
- **Check In**—Check files into the source code control project.

- **Undo Check Out—**Undo check out actions you previously performed on files.

- **Add to Source Control**—Add files to the source code control project.

- **Remove From Source Control**—Remove files from the source code control project.

- **Show History**—View the source code control history for a file.

- **Show Differences**—View the differences between a file on disk and the latest version of the file in the source code control system.

- **Properties**—View the source code control properties and status of a file.

- **Refresh Status**—Have LabWindows/CVI update the source code control status of files. If you use the source code control system GUI to change the status of files in your source code control system, LabWindows/CVI might not know about these changes until you select **Refresh Status**.

- **Clear Source Code Control Error Window**—Clear the contents of the Source Code Control Error window.

- **Source Code Control**—Launch the GUI interface provided by your source code control system.

Tools»Source Code Browser

The Source Code Browser is a cross-reference tool that lists all files, functions, variables, data types, and macros in a program. You can use the browser to identify how different parts of a program interact with each other. The **Identifier/Filename** box lists the last eight items accessed. Browse information is not available in Release configuration.

To access the Source Code Browser, select **Tools»Source Code Browser**, **View»Source Code Browser**, or press <Ctrl-F1> while placing the cursor over functions, variables, data types, and macros in code. You also can enable the **Generate source code browse information** option in the Build Options dialog box. **Generate source code browse information** generates browse information in the Source Code Browser at compile time.

You can find a definition by selecting **Edit»Go to Definition**, go to the next reference by selecting **Edit»Go to Next Reference**, or return to the previous location by selecting **Edit»Go Back**. You also can type a substring or the entire name of the file, function, variable, data type, or macro in the **Identifier/Filename** box. Wildcards are not supported, and the search is not case sensitive. All search matches appear in the **Matches found** box.

Browsing on Files

Under the Views column, you can choose one of the following items:

- **Functions**—The functions defined in the file. The line number where the function exists in the code appears next to the function name. Static functions are identified.

- **Included headers**—A hierarchical display of included files in the file.

- **Macros**—The macros in the file. The line number where the macro exists appears next to the macro name.

- **Referencing**—A hierarchical list of variables and functions, grouped according to module, explicitly referenced in the file.

- **Referenced from**—The functions and variables, grouped according to module, referenced from another file.

- **Types**—The types defined in the file.

- **Variables**—The variables defined in the file. Line numbers appear next to the variable name. Static variables also are identified next to the line number.

Browsing on Functions

Under the Views column, you can choose from the following items:

- **Definition**—The file that contains the definition for the function, along with the line number in the code.

- **References**—A hierarchical list of files where the function is referenced.

- **Calling**—List of functions that the function is calling and the number of references.

- **Called from**—A hierarchical list of functions that call this function.

Browsing on Variables, Data Types, and Macros

Under the Views column, you can choose from the following items:

- **Definition**—A list of file locations where the variable, data type, or macro is defined.

- **References**—The files and line numbers where you can find references to the variable, data type, or macro.

Tools»UI to Code Converter

Use this utility to generate code that programmatically creates the panels and menu bars in the user interface resource (`.uir`) files you select.

The UI to Code Converter is a LabWindows/CVI application that reads the contents of `.tui` files (alternate format `.uir` files) and generates code to build a user interface (UI) programmatically, eliminating the need to distribute a separate file with your LabWindows/CVI application. The UI to Code Converter works best on `.tui` files created with LabWindows/CVI 5.5 or later. For more information about using the converter, refer to the help in the utility.

Tools»User Interface Localizer

The CVI GUI Localization Utility translates user interface resource (`.uir`) files into other languages.

Use this utility to create a language resource (`.lwl`) file that you can use with `LoadLocalizedPanel` to translate a `.uir` file. For more information about translating files, refer to the help in the localization utility.

Tools»Convert UI to Lab Style

Use the **Convert UI to Lab Style** command to convert Classic Style controls to Lab Style controls. Converting to Lab Style controls does not change control settings you have specified.

When you select this command, the UIR Conversion dialog box appears.

• **Original .UIR**—Use the **Browse** button to browse to the `.uir` you want to convert.

• **New .UIR**—Enter the path and name of the new `.uir`. By default, LabWindows/CVI uses the same path and name of the original `.uir`.

• **Backup Original .UIR**—Enable to create a backup of the original `.uir`. LabWindows/CVI creates the backup in the original location.

• **Convert UIR**—Convert the `.uir`.

• **Cancel**—Cancel the operation and close the UIR Conversion dialog box.

User-Defined Entries in the Tools Menu

You can install your own entries in the **Tools** menu. Each entry invokes an executable with optional command line arguments.

Tools»Customize

Use the **Customize** command to add your own menu items to the **Tools** menu. Each entry consists of a menu item name and an associated command line to execute. Each command line consists of a program name and optional arguments. When you execute an item from the **Tools** menu, LabWindows/CVI calls a system function to start the program as another process.

• **Menu Item Name**—Contains your current **Tools** menu entries.

• **Command Line**—Shows the command line for the currently selected menu item name.

• **Add**—Adds a new entry.

• **Edit**—Edits the selected entry.

• **Cut**—Removes the selected entry.

- **Copy**—Copies the selected entry.
- **Paste**—Adds the cut or copied entry to the **Tools** menu.

The **Add** and **Edit** buttons open the Add/Edit Tools Menu Item dialog box, where you can set the following options:

- **Menu Item Name**—The string that appears in the **Tools** menu. To specify a shortcut key for the menu item, insert two underscores in front of any letter. You can select that entry by pressing that letter while the **Tools** menu is open.
- **Program Name**—The pathname of the program to execute when you select the menu entry. You can specify a full pathname, a simple filename, or a relative pathname. You can use the **Browse** button to look for the program on disk.
- **Command Line Arguments**—Arguments you want to pass to the program. You can leave this entry blank.

LabWindows/CVI saves your **Tools** menu entries from one session to another, not in the project.

Window Menu for the Workspace Window

Use the commands in the Workspace window **Window** menu to bring any open window to the front for viewing or editing.

Window»Cascade Windows

Use this command to arrange all open windows within the Window Confinement Region so that each title bar is visible.

This command does not arrange released windows.

Window»Tile Windows

Use this command to arrange all open windows in smaller sizes to fit next to each other in the Window Confinement Region.

This command does not tile released windows.

Window»Minimize All

The **Minimize All** command hides all confined LabWindows/CVI windows.

Window»Close All

Use this command to close all open windows within the Window Confinement Region.

This command does not close released windows.

Window»Workspace

Use this command to change the keyboard focus to the Workspace trees or to activate the Workspace window from a different window.

Window»Build Errors

If you attempt to build a project and the project has build errors, such as syntax or link errors, the Build Errors window contains a list of the errors. To bring the Build Errors window to the front for viewing, select **Window»Build Errors**. You can double-click an error in the Build Errors window to highlight the line of code where the error occurred.

The context menu in the Build Errors window contains the following items:

- **Clear Window**—Removes all items from the Build Errors window.
- **Copy Item**—Copies a single, highlighted item to the clipboard.
- **Copy All Items**—Copies all items in the Build Errors window to the clipboard.
- **Save to File**—Saves the contents of the Build Errors window to a text file.
- **Goto Item**—Highlights the selected line of code.
- **Release/Confine Window**—Releases the window from the Output Window Region if confined. Confines the window into the Output Window Region if released.
- **Hide Confined Error Windows**—Removes all windows from the Output Window Region.

To access the context menu, right-click an empty area of the Build Errors window.

Window»Run-Time Errors

If you attempt to run a project and the project has run-time errors, such as over-indexing an array, the Run-Time Errors window contains a list of the errors. The Run-Time Errors window also displays the output of the Utility Library `ErrorPrintf` function. To bring the Run-Time Errors window to the front for viewing, select **Window»Run-Time Errors**.

The context menu in the Run-Time Errors window contains the following items:

- **Clear Window**—Removes all items from the Run-Time Errors window.
- **Copy Item**—Copies a single, highlighted item to the clipboard.
- **Copy All Items**—Copies all items in the Run-Time Errors window to the clipboard.

- **Save to File**—Saves the contents of the Run-Time Errors window to a text file.

- **Goto Item**—Highlights the selected line of code.

- **Release/Confine Window**—Releases the window from the Output Window Region if confined. Confines the window into the Output Window Region if released.

- **Hide Confined Error Windows**—Removes all windows from the Output Window Region.

To access the context menu, right-click an empty area of the Run-Time Errors window.

Window»Debug Output

LabWindows/CVI displays the output of the Utility Library `DebugPrintf` function and the Windows SDK `OutputDebugString` function in the Debug Output window. If you double-click the output of `DebugPrintf` and `OutputDebugString` in the Debug Output window, LabWindows/CVI highlights the function call in the source code.

Use the `DebugPrintf` function for all your debug output strings. Unlike the Standard I/O window, you can access and scroll through the Debug Output window even while your program is suspended. To view the Debug Output window as the program adds text to the window, enable the **Bring Debug Output Window to Front whenever Modified** option in the Environment dialog box.

The context menu in the Debug Output window contains the following items:

- **Clear Window**—Removes all items from the Debug Output window.

- **Copy Item**—Copies a single, highlighted item to the clipboard.

- **Copy All Items**—Copies all items in the Debug Output window to the clipboard.

- **Save to File**—Saves the contents of the Debug Output window to a text file.

- **Goto Item**—Highlights the selected line of code.

- **Release/Confine Window**—Releases the window from the Output Window Region if confined. Confines the window into the Output Window Region if released.

- **Hide Confined Error Windows**—Removes all windows from the Output Window Region.

To access the context menu, right-click an empty area of the Debug Output window.

Window»Find Results Window

The Find Results window contains the search results for a search involving multiple files. The Find Results window lists the filename and line number of the matched text. Double-clicking a match in the Find Results window opens the file and highlights the matched text. Press <F4> to cycles through the matches (if you have default shortcut keys enabled).

The context menu in the Find Results window contains the following items:

- **Clear Window**—Removes all items from the Find Results window.

- **Copy Item**—Copies a single, highlighted item to the clipboard.

- **Copy All Items**—Copies all items in the Find Results window to the clipboard.

- **Save to File**—Saves the contents of the Find Results window to a text file.

- **Goto Item**—Highlights the selected line of code.

- **Release/Confine Window**—Releases the window from the Output Window Region if confined. Confines the window into the Output Window Region if released.

- **Hide Confined Error Windows**—Removes all windows from the Output Window Region.

To access the context menu, right-click an empty area of the Find Results window.

Window»Source Code Control Errors

The Source Code Control Errors window displays warnings and errors that source code control systems return when you execute commands from the **Tools»Source Code Control** submenu.

The context menu in the Source Code Control Errors window contains the following items:

- **Clear Window**—Removes all items from the Source Code Control Errors window.

- **Copy Item**—Copies a single, highlighted item to the clipboard.

- **Copy All Items**—Copies all items in the Source Code Control Errors window to the clipboard.

- **Save to File**—Saves the contents of the Source Code Control Errors window to a text file.

- **Release/Confine Window**—Releases the window from the Output Window Region if confined. Confines the window into the Output Window Region if released.

- **Hide Confined Error Windows**—Removes all windows from the Output Window Region.

To access the context menu, right-click an empty area of the Source Code Control Errors window.

Window»Memory Display

The **Memory Display** command opens the Memory Display window in which you can view and edit the memory of the program you are debugging. Use this window to view and edit the data in hexadecimal (byte, word, long), decimal (byte, word, long), single-precision floating point, double-precision floating point, or ASCII representation. You must enable the **Edit**

Mode option before you can modify the process memory. To change the value of a memory location, click that cell in the Memory Display window and type the new value.

You can drop variables onto the Memory Display window. To drop variables onto the Memory Display window, select the variable you want to view from the Source, Interactive Execution, Variables, or Watch window and drop it onto the Memory Display window.

Window»Variables

Use this command to bring the Variables window to the front. The Variables window shows the contents of all variables currently defined in the program. You can access the Array Display and String Display windows from the Variables window.

The Variables window is useful for debugging programs. LabWindows/CVI updates the Variables window at each breakpoint, and you can modify the variables while in a breakpoint state.

Window»Watch

The **Watch** command brings the Watch window to the front. The Watch window shows a set of variables and expressions that you specify. You can access the Array Display and String Display windows from the Watch window.

The Watch window is useful for debugging programs. Watch variables and expressions update at each breakpoint unless you set them to update continuously.

Window»Array Display

If an Array Display window is active, it appears in the **Window** menu. Selecting an Array Display window from the **Window** menu brings it to the front for viewing and editing.

Window»String Display

If a String Display window is active, it appears in the **Window** menu. Selecting a String Display window from the **Window** menu brings it to the front for viewing and editing.

Window»User Interface

All open user interface resource (`.uir`) files dynamically appear in the **Window** menu. If the file is in the Window Confinement Region, only the filename appears. If the file is not in the Window Confinement Region, the full pathname appears. Select a `.uir` file from the **User Interface** submenu to bring the corresponding User Interface Editor window to the front.

Window»Function Panel

All open Function Panel windows dynamically appear in the **Window** menu. Select a window from the **Function Panel** submenu to bring that window to the front.

Window»Function Tree

All open Function Tree files dynamically appear in the **Window** menu. If the file is in the Window Confinement Region, only the filename appears. If the file is not in the Window Confinement Region, the full pathname appears. Select a `.fp` file from the **Function Tree** submenu to bring the corresponding Function Tree Editor window to the front.

Window»Help Editor

All open Help Editor windows dynamically appear in the **Window** menu. Select a window from the **Help Editor** submenu to bring that Help Editor window to the front.

Window»Interactive Execution

Use the **Interactive Execution** command to bring the Interactive Execution window to the front. Unlike the Source window, you can execute incomplete programs in the Interactive Execution window. For example, you can execute variable declarations and assignment statements in C without declaring a `main` function.

Window»Open Source Files

The **Window** menu lists open source files at the bottom. If the file is in the Window Confinement Region, only the filename appears. If the file is not in the Window Confinement Region, the full pathname appears. Select a source file from this menu to bring its window to the front.

Options Menu for the Workspace Window

Use commands in the **Options** menu to set preferences in the LabWindows/CVI environment.

Options»Environment

The **Environment** command opens the Environment dialog box you can use to set the following options:

- **Hide Windows**—Hides all LabWindows/CVI windows until execution terminates or a breakpoint occurs.

- **Use Only One Browse Info Window**—Overwrites the current browse info window each time you select a new file, function, variable, data type, or macro.

- **Save Changes before Debugging**—Sets LabWindows/CVI to never save modified files, to always save modified files, or to ask whether to save modified files before debugging.

- **Lines in Debug Output Window**—Shows the last *n* number of lines that you specify.

- **Bring Debug Output Window to Front whenever Modified**—Allows LabWindows/CVI to bring the Debug Output window to the front whenever your program adds text to the window.

- **Use Console Window for Standard I/O when Debugging**—Allows LabWindows/CVI to use the Microsoft DOS console window for displaying output and receiving input while you debug a project.

- **Copy Standard I/O to Debug Output Window**—Copies the contents of the Standard I/O window to the Debug Output. You must enable this option prior to debugging in order for LabWindows/CVI to copy the contents to the Debug Output window.

- **Enable Data ToolTips**—Allows LabWindows/CVI to display the values of variables and selected expressions when you place the mouse cursor over the variables or selected expressions in a Source window.

- **Enable Global Ctrl+F12 Debug Break Key**—Enables a system-wide hot key to suspend the execution of your LabWindows/CVI program or the Interactive Execution window. You can use this hot key when LabWindows/CVI or your program is not the active application. Enabling this option might interfere with other applications (including your program) that use <Ctrl-F12> for different purposes. The hot key also might not work correctly when you are debugging two applications with LabWindows/CVI at the same time. The hot key is active only when LabWindows/CVI is running a user program or the Interactive Execution window.

- **Interactive Window Memory Size**—Sets the amount of memory you want LabWindows/CVI to reserve for executing function panels or code in the Interactive Execution window. If LabWindows/CVI displays the message `Insufficient memory for Interactive window`, you must increase the value of **Interactive Window Memory Size**, select **Build»Clear Interactive Declarations** in the Interactive Execution window, and then execute the function panel or Interactive Execution window statements again.

- **Force Project Files into Interactive Window**—Enable this option if you run function panel or Interactive Execution window code that uses `LoadExternalModule` or `LoadExternalModuleEx` to load a source file that is in the project.

- **Force Loaded Instrument Drivers into Interactive Window**—Enable this option if you run function panel or Interactive Execution window code that uses `LoadExternalModule` or `LoadExternalModuleEx` to load the source file for an instrument driver that is already loaded in the **Instrument** menu.

- **CVI Environment Sleep Policy**—Each time LabWindows/CVI checks an event from the operating system, it can put itself in the background, in sleep mode, for a specified period of time. While LabWindows/CVI is in sleep mode, other applications have more

processor time. However, LabWindows/CVI might run slower. You can specify how much LabWindows/CVI sleeps. You have the following sleep policy choices:

– **Do Not Sleep**

– **Sleep Some** (sleep a short period of time)

– **Sleep More** (sleep a longer period of time, the default setting)

The setting that is optimal for you depends on the operating system you are using and the other applications you are running. Generally for Windows, National Instruments recommends the **Sleep More** mode. If you think you might need to make an adjustment, try the different settings and observe the resulting behavior.

• **Include Paths**—Opens the Include Paths dialog box, in which you can specify the directory search path for include files.

• **Source Code Control**—Opens the Source Code Control Options dialog box, in which you can select settings for your source code control system.

Include Paths

The **Include Paths** option in the Environment dialog box invokes a dialog box in which you can list paths that the compiler uses when searching for header files with simple pathnames. The Include Paths dialog box has two lists of paths in which to search for include files, one for include paths specific to the project and one for paths not specific to the project. LabWindows/CVI saves the list of paths specific to the project with the project file. LabWindows/CVI saves the list of paths not specific to the project from one session to another on the same machine, regardless of the project. When you install VXI*plug&play* instrument drivers, the installation program places the include files for the drivers in a specific VXI*plug&play* `include` directory. If you install IVI instrument drivers, the installation program places the include files for those drivers in a specific IVI `include` directory. LabWindows/CVI also searches those directories for include files.

Source Code Control Options

The **Source Code Control** option in the Environment and Edit Project dialog boxes opens the Source Code Control Options dialog box, where you can set the following options:

• **Provider**—The name of the source code control system that contains the source code control project.

• **Project**—The name of the source code control project.

• **Attach**—Attach an existing source code control system project. After you attach a source code control provider, this button changes to **Change**. Use **Change** to change source code control system projects.

• **Create**—Create a new source code control system project.

- **Perform same actions for .h file as for .uir file**—Because LabWindows/CVI generates a header file for the .uir file each time you save the .uir file, it is a good idea to perform the same source code actions on the header file as you perform on the .uir file.

  - **Never**—When you perform a source code control action, LabWindows/CVI does not perform the same action on the header file and on the associated .uir file.

  - **Ask**—When you perform a source code control action, LabWindows/CVI asks if you want to perform the same action on the header file and on the associated .uir file.

  - **Always**—When you perform a source code control action, LabWindows/CVI automatically performs the same action on the header file and on the associated .uir file.

- **Suppress CVI Error Message dialog**—Some source code control systems display their own dialog boxes when errors occur during a source code control operation. Enabling this option suppresses all LabWindows/CVI error dialog boxes displayed when an error is reported by the source code control system.

- **Always show confirmation dialog**—Displays a dialog box where you can confirm your source code control actions.

- **Use default comment**—Enable this option if you do not want to be prompted for a comment when you check in or add files to a source code control project. Enter the comment that you want LabWindows/CVI to pass to the source code control system.

- **Use default username**—You can change your username by enabling this option and entering a new name.

- **Advanced**—Access advanced options provided by your source code control system. If your source code control system does not provide advanced options, this control is dimmed.

Options»Build Options

You can set the LabWindows/CVI compiler options by selecting **Options»Build Options** in the Workspace window. This command opens a dialog box in which you can set the following LabWindows/CVI compiler options:

- **Compatibility with**—This option displays the current compiler compatibility mode. You can change the current compatible compiler by selecting **Visual C/C++** or **Borland C/C++**. You must restart LabWindows/CVI after changing the compiler compatibility.

- **Default calling convention**—This option sets the compiler's default calling convention. For both compilers, the default calling convention is normally __cdecl but can be changed to __stdcall. Do not change the default calling convention to __stdcall if you plan to generate static library or object files for both compatible external compilers.

Note   If you want to create an object file, static library file, or DLL that exports functions with the __stdcall calling convention, it is a good idea to explicitly declare the functions

as `__stdcall` in the include (`.h`) file and the source (`.c`) file rather than relying on the **Default calling convention** option. If you do not explicitly declare the functions as `__stdcall` in the include file and if another developer uses the object file, library file, or DLL in LabWindows/CVI or an external compiler without setting the default calling convention to `__stdcall`, the functions do not work correctly.

- **Maximum stack size (bytes)**—Your program uses the stack for passing function parameters and storing automatic local variables. By setting a stack limit, LabWindows/CVI can catch infinitely recursive functions and report a stack overflow. If you enable the **Detect uninitialized local variables at run time** option, LabWindows/CVI uses extra stack space. You can adjust your maximum stack size to avoid a stack overflow.

- **Image base address**—This option specifies the address where LabWindows/CVI loads a DLL or executable into memory. By specifying the image base address, you can avoid relocating DLLs, which can slow down the load time of DLLs. You also can avoid collisions, which occur when LabWindows/CVI attempts to load more than one DLL with the same address.

  You can specify any value, or you can select the following default values:

  – x00400000 for executables

  – x10000000 for DLLs

- **Debugging level**—LabWindows/CVI uses this setting only if you enable the **Build»Configuration»Debug** item of the Workspace window. If you check the **Release** item in the **Configuration** submenu, LabWindows/CVI compiles all source files without debugging information.

  Select one of the three debugging levels for the source modules in your application.

  – **No run-time checking**—In this mode, you can set breakpoints and use the Variables window. You have no protection from run-time memory errors, and you cannot use the **Run»Break on»Library Errors** option.

  – **Standard**—In this mode, you can set breakpoints, use the Variables window, and use the **Run»Break on»Library Errors** option. You also have protection from run-time memory errors.

  – **Extended**—This mode has the same benefits as Standard mode, with added user protection that validates every attempt to free dynamically allocated memory by verifying that the address you pass is actually the beginning of an allocated block.

- **Detect uninitialized local variables at run time**—This option checks for the run-time use of variables that do not have values assigned to them. Enabling this option causes LabWindows/CVI to use extra stack space. To avoid a stack overflow, adjust the maximum stack size.

- **Track include file dependencies**—This option keeps the project up-to-date by tracking the dependencies between source files and include files. Whenever you modify a file, LabWindows/CVI marks for compilation all source files that include the modified file.

- **Prompt for include file paths**—This option sets LabWindows/CVI to prompt you to make a manual search for any header files listed in the `#include` lines that the compiler cannot find. When you find them, you can automatically insert the appropriate path into the Include Paths list for the project.

- **Display status dialog during build**—This option displays a status dialog box during the build that shows the name of the file being compiled, the number of errors and warnings encountered, and a percent completed value. The project compiles faster when you disable this feature.

- **Make 'O' option compatible with CVI 5.0.1**—When you enable this option, the 'O' option performs as it did in LabWindows/CVI 5.0.1. LabWindows/CVI compiles the source file without debugging information and writes the `.obj` file to disk in the same directory as the source file. To set a file with the 'O' option, right-click a file in the Project Tree and select **Enable 'O' Option** from the context menu.

- **Uninitialized local variables detection**—This option generates error messages when you access variables that do not have values assigned to them. You can select the following levels:

  - **Disabled**—This mode disables uninitialized local variable warnings and errors.

  - **Conservative**—This mode flags only variables that do not have values assigned to them.

  - **Aggressive**—This mode flags all variables that may or may not have values assigned to them.

- **Detect signed/unsigned pointer mismatches**—This option generates a compiler warning for pointer assignments in which the left side and right side are not both `signed` or `unsigned` expressions. According to the ANSI C standard, these assignments are errors because they involve incompatible types. In practice, however, such assignments cause no problems.

  The LabWindows/CVI compiler checks assignment statements and function call arguments to ensure that the `lvalue` and `rvalue` expressions have compatible types. If you enable this option, LabWindows/CVI generates compile warnings when the `lvalue` and `rvalue` expressions are both pointers to integers but one points to a signed integer and the other points to an unsigned integer. For example, the LabWindows/CVI compiler generates a **signed type mismatch between pointer to char and pointer to unsigned char** warning on the call to `MyFunction` in the following code example:

```
void MyFunction (unsigned char *x);
char *y = "my string";
main () {
```

```
    MyFunction (y);
}
```

- **Detect unreachable code**—This option generates a compiler warning for statements that the compiler cannot reach on execution. When you enable this option, the LabWindows/CVI compiler generates a warning at each line of code that cannot be reached during the execution of your program. For example, LabWindows/CVI reports a warning on the break statement in the following code:

```
switch (intval) {
    case 4:
    return 0;
    break;
}
```

- **Detect unreferenced identifiers**—This option generates compiler warnings for identifiers that are not referenced in your program.

- **Detect assignments in conditional expressions**—LabWindows/CVI checks for possible errors in conditional expressions that can make the conditional expression an assignment.

- **Require function prototypes**—This option requires you to precede all function references with a full prototype declaration. A full prototype includes the function return type and the types of each parameter. If a function has no parameters, a full prototype must have the void keyword to indicate this case. A new style function definition (one in which you declare parameters directly in the parameter list) can serve as a prototype.

  Missing prototype errors can occur at the following places:

  – Typedefs such as `typedef void FUNTYPE()`

  – Function pointer declarations such as `void (*fp)()` whether used as a global, local, parameter, array element, or structure member

  – Old style function definitions, in which you declare parameters outside of the parameter list, that you do not precede with a full prototype

  – Function call expressions such as `(*fp)()`, where `fp` does not have a full prototype

✎ **Note**   It is best to enable the **Require function prototypes** option. If disabled, some of the run-time error checking also is disabled.

- **Require return values for non-void functions**—This option generates compile warnings for non-void functions, except `main`, that do not end with a `return` statement that returns a value. LabWindows/CVI reports a run-time error when a non-void function executes without returning a value.

For example, the following code always produces a compile-time warning, and it produces a run-time error when `flag` is `FALSE`.

```
int fun (void)
{
    if (flag) {
    return 0;
    }

}
```

- **Maximum number of compile errors**—This option sets an upper limit on the number of compile errors that LabWindows/CVI lists in the Build Errors window for each source file.

- **Stop on first file with errors**—This option sets the LabWindows/CVI compiler to terminate compilation after it finds one file with errors. Using this option, you can correct build errors in your project one file at a time.

- **Show build error window for warnings**—This option sets the LabWindows/CVI compiler to open the Build Error window when warnings occur, even if no errors exist. If you deactivate this option, warnings can occur without being brought to your attention.

- **Generate source code browse information**—Use this option to view source code browse information under **Tools»Source Code Browser**.

- **Compiler Defines**—Use this option to set compiler defines.

- **Predefined Macros**—This option lists the predefined macros.

Compiler Defines

The LabWindows/CVI compiler accepts compiler defines in the Build Options dialog box, accessible through **Options»Build Options**.

Compiler defines have the following syntax:

`/D`x or `/D`x`=`y

The variable x is a valid C identifier. You can use x in source code as a predefined macro. For example, you can use x as the argument to the `#if` or `#ifdef` preprocessor directive for conditional compilation. If y contains embedded blanks, you must surround it with double quotation marks.

The Predefined Macros dialog box contains a list of the macros that LabWindows/CVI predefines. This list includes the name and value of each predefined macro.

Note The default compiler defines string contains the following definition: `/DWIN32_LEAN_AND_MEAN`. This definition reduces the time and memory taken to compile Windows SDK include files.

Predefined Macros

- _CVI_ is defined to 1 in LabWindows/CVI version 3.0, 301 in version 3.0.1, 310 in version 3.1, and so on.

- _NI_mswin_

- _NI_mswin32_

- _NI_i386_

- _CVI_DEBUG_ is defined if you enable **Build»Configuration»Debug** in the Workspace window. The value of the macro is 1.

- _CVI_EXE_ is defined if the project target type is **Executable**.

- _CVI_DLL_ is defined if target type is **Dynamic Link Library**.

- _CVI_LIB_ is defined if target type is **Static Library**.

- _LINK_CVIRTE_ is defined if the project run-time support is **Full Run-Time Engine**. The target will link to cvirte.dll.

- _LINK_INSTRSUP_ is defined if the project run-time support is **Instrument Driver Only**. The target will link to instrsup.dll.

- _LINK_CVI_LVRT_ is defined if the project run-time support is **LabVIEW Real-Time Only**. The target will link to cvi_lvrt.dll.

- __DEFALIGN is defined to the default structure alignment: 8 for Microsoft; 1 for Borland.

- _NI_VC_ is defined to 220 if in Microsoft Visual C/C++ compatibility mode.

- _NI_BC_ is defined to 451 if in Borland C/C++ compatibility mode.

- _WINDOWS

- WIN32

- _WIN32

- __WIN32__

- __NT__

- _M_IX86 is defined to 400.

- __FLAT__ is defined to 1.

- _CVI_USE_FUNCS_FOR_VARS_

- _PUSHPOP_SUPPORTED

- LabWindows/CVI does not define _MSC_VER or __BORLANDC__. The external compilers each define one of these macros. If you port code originally developed under one of these external compilers to LabWindows/CVI, you might have to manually define one of these macros.

Options»Change Shortcut Keys

Use this dialog box to reassign shortcut keys.

- **Window**—Contains all of the windows in the LabWindows/CVI environment.

- **Current Shortcut Keys**—Lists the currently assigned shortcut keys for each command in the window you selected.

- **Reset All**—Sets the default assignments for all shortcuts in all windows.

- **Load**—Loads a shortcut key list.

- **Save**—Saves to disk the currently assigned shortcut keys.

- **OK**—Accepts changes you made and closes the dialog box.

- **Cancel**—Discards changes you made and closes the dialog box.

- **Help**—Opens the help topic for this dialog box.

Right-click a menu command in Current Shortcut Keys to access the context menu.

- **Change Key**—Opens the Change Key dialog box. The Change Key dialog box contains the following options:

  - **Modifier Key**—Select **None**, **Ctrl**, **Shift**, or **Ctrl+Shift**.

  - **Shortcut Key**—What you select in **Modifier Key** determines what appears in **Shortcut Key**. You can select function keys, letter keys, arrow keys, and space keys.

  - **Free Keys**—View a list of available keyboard shortcuts. Select a shortcut you want to use and click **Use Key**.

    If you assign a shortcut key that is already in use, a Conflicting Keys message appears, which shows the window and command assigned to the shortcut you are attempting to use. You can select **Clear the Conflicting Keys**, which removes the currently assigned shortcuts from all windows in which the conflicting shortcut appears and accepts the new shortcut, or you can **Revert** to the shortcut that was assigned previously.

- **Reset key**—Sets the default shortcut key for the command.

- **Find**—Searches through the shortcut keys for a specified string.

- **Expand All**—Opens all items and subitems in the shortcut key list.

- **Collapse All**—Closes all items and subitems in the shortcut key list.

Options»Colors

Use the **Colors** menu item to select colors for the Workspace window, Source window, Interactive Execution window, Standard I/O window, Watch window, Variables window, String Display window, and Array Display window. The **Colors** menu item does not affect dialog boxes, function panels, and the User Interface Editor.

The **Colors** menu item opens the Environment Color Preferences dialog box. The dialog box contains the purpose of the color and shows its current color state. To change the color, select an item in the list and click the colored box next to **Color**. A color pop-up palette appears. Move the mouse pointer over the color you want and click the color. The color change takes effect immediately in all LabWindows/CVI windows that are currently visible.

To change all the colors to their default state, click **Default**. All currently visible LabWindows/CVI windows immediately reflect the color changes.

If you want to accept changes you made, click **OK**. If you want to revert to the state before the dialog box appeared, click **Cancel**.

The Environment Color Preferences dialog box also contains eight color types for syntax coloring.

When you enable the **Use System Colors** option, several color types associated with the Workspace window, Source window, and scroll bars disappear from the list box. LabWindows/CVI automatically assigns colors to these types based on the system colors you set in the **Appearance** tab in the Windows Display Properties dialog box.

Toolbars in LabWindows/CVI

The LabWindows/CVI toolbar appears within the Workspace window, in Source windows, in the Interactive Execution window, in function panels, and in Function Panel Editor windows. Using the toolbar gives you quick access to common commands, such as **File»Open** and **File»Save**. You can configure the toolbar to meet your needs or choose not to display it at all.

To find out what a toolbar button does, position the mouse cursor over that button and either hold the cursor there for a short period of time or right-click the toolbar button to display the name of the toolbar button.

Items in the toolbar change depending on what windows are open.

Modifying Your Toolbars

To modify a toolbar, select **Options»Toolbar** to open the Customize Toolbars dialog box.

Toolbar contains the windows or combination of windows in which toolbars appear. Select a window to modify its toolbar. The **Available Buttons** list contains names and icons of toolbar buttons that do not currently appear in the toolbar. The **Buttons on Toolbar** list contains the names and icons of toolbar buttons that currently appear in the toolbar.

Adding and Positioning Buttons

Use the Add Button controls to add and position new buttons on the toolbar. First, select the button you want to add to the toolbar from the **Available Buttons** list. In the **Buttons on Toolbar** list, select the button you want to place the new button next to. The **Above** button positions the item you are adding above the button that you selected in the **Buttons on Toolbar** list. After you click **OK**, the new item appears to the left of the other button in the toolbar. The **Below** button positions the item you are adding below the button that you selected in the **Buttons on Toolbar** list. After you click **OK**, the new item appears to the right of the other button in the toolbar.

Adding and Positioning Separators

Use the Add Separator controls to add and position separators on the toolbar. Select a button in the **Buttons on Toolbar** list. The **Above** button adds a separator above the button that you selected in the **Buttons on Toolbar** list. After you click **OK**, a small gap (the separator) appears in the toolbar to the left of the selected button. The **Below** button adds a separator below the button that you selected in the **Buttons on Toolbar** list. After you click **OK**, a small gap (the separator) appears in the toolbar to the right of the selected button.

Other Positioning Controls

You can select any item in the **Buttons on Toolbar** list and use the **Remove** button to remove it from the toolbar. If you remove a button, LabWindows/CVI moves the button to the **Available Buttons** list. If you remove a separator, it disappears from the list. Your modifications take effect on the toolbar when you click **OK**.

Click **Default** to restore the default toolbar configuration for the selected window.

To position any item on the toolbar, select it in the **Buttons on Toolbar** list and click **Move Up** or **Move Down**. Modifications take effect on the toolbar when you click **OK**.

Help Menu for the Workspace Window

Use the commands in the **Help** menu to access information about LabWindows/CVI.

Help»Contents

The **Contents** command opens the *LabWindows/CVI Help*.

Help»Windows SDK

The **Windows SDK** command opens online help for the Windows API functions.

Help»LabWindows/CVI Bookshelf

This command launches the *LabWindows/CVI Bookshelf*, which contains a searchable, comprehensive list of all LabWindows/CVI documentation.

Help»Workspace View Selection

The **Workspace View Selection** command opens a dialog box in which you can change the display of the Workspace window.

Help»Tip of the Day

The **Tip of the Day** command opens a dialog box containing tips to help you learn about features in the LabWindows/CVI environment.

Help»NI Example Finder

This command launches the National Instruments Example Finder. With the NI Example Finder, you can search through all examples that ship with the NI products you have installed on your machine. You can specify various options in your search, such as supported hardware and example functionality, and whether you want to search examples on NI Developer Zone at `ni.com/zone`. You also can submit your own examples to NI Developer Zone.

Refer to the *National Instruments Example Finder Help* for more information.

Help»Web Links

The **Web Links** command has a submenu that contains links to helpful National Instruments Web sites.

Help»Patents

This command displays a dialog box in which you can view a list of patents for National Instruments products.

Help»About LabWindows/CVI

The **About LabWindows/CVI** command displays a read-only dialog box with information about LabWindows/CVI.

3

User Interface Editor

A LabWindows/CVI Graphical User Interface (GUI) can consist of panels, command buttons, pull-down menus, graphs, strip charts, knobs, meters, and other controls and indicators.

You can create a GUI programmatically using function calls, or you can build a GUI in LabWindows/CVI interactively using the User Interface Editor, a drop-and-drag editor with tools for designing, arranging, and customizing user interface objects. With the interactive User Interface Editor, you can build an extensive GUI for your program without writing a single line of code. When you finish designing your GUI in the User Interface Editor, save the GUI as a user interface resource (.uir) file. This section describes the User Interface Editor, tells you how to create a GUI interactively, and explains procedures for creating and editing panels, controls, and menu bars.

When you use the User Interface Editor, you create and modify user interface resource (.uir) files. To open the User Interface Editor, select **File»New** or **File»Open** and select **User Interface (*.uir)**.

User Interface Editor Overview

From the User Interface Editor, you can create and edit GUI panels, controls, and menu bars. Panels in the User Interface Editor contain grid lines that you can use to align and resize controls. You can use the mouse to edit in four different modes. When you click a particular mode tool, the mouse cursor changes to reflect the new editing mode.

LabWindows/CVI provides commonly used control configurations as custom controls. To access these controls, select **Create»Custom Controls**. Besides an Ok and Quit button, you can use several Toolslib controls such as the file browser, path control, and the hot ring control. Toolslib controls have code associated with them, making the controls fully functional without additional code.

Use the following tools in the User Interface Editor.

Use the operating tool to operate objects. When you are in operate mode, events display in the upper right side of the User Interface Editor, under the menu bar. These event displays have a built-in delay to give you time to see each event. Panel grid lines do not appear in operate mode.

Use the editing tool to select, position, and size objects.

Use the labeling tool to modify text associated with objects.

Use the coloring tool to color objects. Right-click the panel to view a color palette from which you can choose a color. Clicking the panel automatically colors the object with the last color you selected in the color palette.

Hold down the <Ctrl> key on the panel to change the coloring tool to an eyedropper tool. When you click an object with the eyedropper tool, the current color of the tool becomes the color of that object. Then you can apply that object's color to another object.

Using the Context Menus of the User Interface Editor

To open a context menu, right-click the User Interface Editor. The type of context menu that appears depends on the surface you click. The following list describes the areas in the User Interface Editor that open context menus.

- If you click the User Interface Editor background, a context menu appears with options to create a panel or a menu bar.

- If you click a panel background, a context menu appears with options to create each of the control types available in LabWindows/CVI.

- If you click a control, a context menu appears with commands with the following options:

    – **Generate Control Callback**—Generates the #include statements and the function skeleton for the selected control and places the code in the target file.

    – **View Control Callback**

    – **Generate ActiveX Control Driver**—Launches the ActiveX Controller Wizard. This option is available only for ActiveX controls.

    – **Generate Custom Control Code**—Generates code for the custom control. This option applies only to custom controls with code associated with them. This option inserts the code from the template file associated with the custom control and places the code under the call to LoadPanel. LabWindows/CVI adds the program file associated with the control to the project.

    – **Control Help**—Opens the *LabWindows/CVI Help* to the control overview for the selected control.

–   **Edit Control**—Opens the Edit Control dialog box.

–   **Properties**—Opens the property sheets for the selected control. This option is available only for ActiveX controls.

CodeBuilder Overview

With LabWindows/CVI CodeBuilder, you can automatically create complete C code that compiles and runs based on a user interface resource (.uir) file you create or edit. Use options in the **Code** menu to produce skeleton code. Skeleton code is syntactically and programmatically correct code that compiles and runs before you type a single line of code. With the CodeBuilder feature, you save the time of typing standard code included in every program, eliminate syntax and typing errors, and maintain an organized source code file with a consistent programming style. Because a CodeBuilder program compiles and runs immediately, you can develop and test the project you create, concentrating on one function at a time.

When you choose **Code»Generate»All Code**, LabWindows/CVI places the #include statements, variable declarations, callback function skeletons, and main function in the source code file you specify as the target file. Each function skeleton contains a switch construct with a case statement for every default event you specify. You can set default events for control callback functions and panel callback functions by choosing **Code»Preferences**. Although skeleton code runs, you must customize it to implement the actions you want to take place for each event.

When you generate code for a specific control or panel callback function, LabWindows/CVI places the skeleton code for that function in the target file in the same complete format used for the **Code»Generate»All Code** command. However, this code might not run. In order for a project to run, a main function must exist. If you lack the main function or any of the callback functions you reference in the .uir file, the code is incomplete.

It is good practice to use the **Code»Generate»All Code** option first to produce a running project from the current state of the .uir file. Then, after adding panels, controls, or menu items to the .uir file, select **Code»Generate»Panel Callback, Control Callbacks**, or **Menu Callbacks**, to make corresponding additions to the source file.

Also with CodeBuilder, you can make sure that your automatically generated program terminates properly. For a CodeBuilder program to terminate successfully, you must include a call to QuitUserInterface. When you choose **Code»Generate»All Code**, the Generate All Code dialog box prompts you to choose which callback functions terminate the program. You can select one or more callback functions to ensure proper program termination.

File Menu for the User Interface Editor

This section describes how to use the commands in the User Interface Editor **File** menu.

File»New, Open, Save, Save All, Most Recently Closed Files, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, **Save All**, **Most Recently Closed Files**, and **Exit LabWindows/CVI** commands in the **File** menu of the User Interface Editor work like the commands in the **File** menu of the Workspace window. For more information about these commands, refer to the *File Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

File»Save As

Use the **Save As** command to write the file to disk using a name you specify. The **Save As** command changes the name on the title bar to the new name you specified.

File»Save Copy As

The **Save Copy As** command writes the contents of the active window to disk using a name you specify without changing the name of the active window.

File»Close

The **Close** command closes the active window. If you modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

File»Add File to Project

The **Add File to Project** command adds the file in the current window to the project list of the active project.

File»Read Only

The **Read Only** command suppresses the editing capabilities in the current window. When you initially open a file, the **Read Only** command is disabled unless the file is read-only on disk.

File»Print

The **Print** command opens the Print dialog box in which you can send the entire .uir file or the visible screen area to a printer or a file. You also can set print preferences in the Print dialog box. The print preferences correspond to the print attributes described in the *User Interface Library* section of the *LabWindows/CVI Help*.

Edit Menu for the User Interface Editor

This section explains how to use the commands in the User Interface Editor **Edit** menu. Use the **Edit** menu to edit panels, controls, and menu bars.

Edit»Undo and Redo

✎ **Note**   **Undo** and **Redo** are enabled when you perform an edit action. **Cut** and **Copy** are enabled when you select a control. **Paste** is enabled when you place an object on the clipboard using the **Cut** or **Copy** command. If you select an edit command while it is disabled, nothing happens.

The **Undo** command reverses your last edit action, and the screen returns to its previous state. Edit actions are stored on a stack so that you can undo a series of your edit actions. The stack can store up to 100 edit actions. To set the size of the undo stack, select **Options»Preferences** and click **More**.

The **Redo** command reverses your last **Undo** command, restoring the screen to its previous state. **Redo** is helpful when you use the **Undo** command to reverse a series of your edit actions and accidentally go too far. The **Redo** command is enabled only when your previous action was the **Undo** command. Any other action disables the **Redo** command.

Actions that you can undo and redo appear dynamically in the menu. For example, when you move a control, the menu presents the **Undo Move Control** option.

Edit»Cut and Copy

To cut or copy controls to the clipboard, select the control you want to place on the clipboard and then select **Edit»Cut** or **Copy**. LabWindows/CVI places the selected control and its associated help on the clipboard. If you used the **Copy** command, the control remains in the window. Use the **Cut** command to delete controls from the window. Controls you cut or copy do not accumulate on the clipboard. LabWindows/CVI replaces a control every time you cut or copy a control to the clipboard.

To use the **Cut** or **Copy** commands, complete the following steps:

1. Select the control you want to place on the clipboard by clicking the control or pressing <Tab> until the control is highlighted. Select multiple controls by dragging the mouse over the controls. You also can press <Shift-Click> to select multiple controls.

2. Select **Edit»Cut** or **Copy**.

Edit»Paste

The **Paste** command inserts controls, panels, or text from the clipboard. You can paste an object from the clipboard as many times as you like. Controls or panels remain on the clipboard until you use **Cut**, **Cut Panel**, **Copy**, or **Copy Panel** again. The **New** and **Open** commands do not erase the clipboard.

Edit»Delete

The **Delete** command deletes selected controls without placing the controls on the clipboard. Because **Delete** does not place controls on the clipboard, you cannot restore deleted controls using the **Paste** command.

Edit»Copy Panel and Cut Panel

The **Copy Panel** and **Cut Panel** commands put an entire panel on the clipboard. The **Copy Panel** command copies the selected panel, its controls, and all the associated help information and places it on the clipboard, leaving the selected panel in its original location. The **Cut Panel** command removes the selected panel, its controls, and all the associated help information and places it on the clipboard. Panels you cut or copy do not accumulate on the clipboard. LabWindows/CVI replaces an existing panel every time you cut or copy a panel.

To use the **Copy Panel** or **Cut Panel** commands, follow these steps:

1. Select the panel you want to place on the clipboard by clicking the panel or pressing <Shift-Ctrl> and the left or right arrow key until the panel is highlighted.

2. Select **Edit»Copy Panel** or **Cut Panel**.

Edit»Menu Bars

The **Menu Bars** command opens the Menu Bar List dialog box.

The list contains all of the menu bars in the resource file, listed by constant prefix. The Menu Bar List dialog box has the following options:

*   **Create**—Opens a new Edit Menu Bar dialog box. After you create a menu bar, it appears below the currently selected menu bar in the menu bar list.

*   **Edit**—Opens the Edit Menu Bar dialog box for the selected menu bar.

*   **Cut**—Deletes the currently highlighted item in the menu bar list and copies it to the menu bar clipboard.

*   **Copy**—Copies the currently highlighted item in the menu bar list to the menu bar clipboard.

- **Paste**—Inserts the contents of the menu bar clipboard to the menu bar list. When you use the **Paste** button, the menu bar is inserted above the currently highlighted item in the menu bar list.

- **Done**—Closes the Menu Bar List dialog box.

Edit Menu Bar

The Edit Menu Bar dialog box contains the following options:

- **Menu Bar Constant Prefix**—Sets the resource ID for the menu bar. Pass this resource ID to `LoadMenuBar` to load the menu bar into memory. The menu bar constant prefix is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a menu bar constant prefix, the User Interface Editor assigns one for you when you save the `.uir` file.

- **Constant Name**—Sets the constant name of the item, which LabWindows/CVI appends to the menu bar constant prefix to form the ID for the current item. Pass the ID to functions such as `GetMenuBarAttribute` and `SetMenuBarAttribute`. `GetUserEvent` returns the ID when the current menu item generates a commit event.

- **Item Name**—Sets the name of the current menu, submenu, or menu command. If you type a double underscore before any letter in **Item Name**, the letter appears underlined in the label. The user can select the menu item by pressing <Alt> and that letter.

- **Callback Function**—This field is optional. Type the name of the function to call when the current menu item generates an event.

- **Modifier Key**—Identifies the keys that users can press to execute the current menu item.

- **Shortcut Key**—Identifies the keys that users can press to execute the current menu item.

- **Dimmed**—Specifies whether the menu item is initially dimmed.

- **Checked**—Specifies whether the menu item initially has a checkmark.

- **Bold**—Specifies whether the menu item is initially bold.

- **Insert Item**—Inserts a new item below the currently selected menu item.

- **Insert Child Item**—Inserts a new sub-item below the currently selected menu item.

- **Insert Separator**—Inserts a separator below the currently selected menu item.

- **Move Up**—Moves the currently selected item up one item in the hierarchy.

- **<< Change Level**—Moves the currently selected menu item up one level in the menu hierarchy.

- **Change Level >>**—Moves the currently selected item down one level in the menu hierarchy.

- **Move Down**—Moves the currently selected item down one item in the menu hierarchy.

- **Cut**—Deletes the currently selected menu item and copies it to the menu clipboard.

- **Copy**—Copies the currently selected menu item to the menu clipboard.

- **Paste**—Inserts the menu item currently on the menu clipboard below the currently selected menu item.

- **Paste Child**—Inserts the item currently on the menu clipboard as the child of the currently selected menu item.

- **OK**—Accepts the current inputs and closes the dialog box.

- **Cancel**—Cancels the operation and removes the dialog box.

- **Help**—Displays the help topic for the Edit Menu Bar dialog box.

- **Preview**—Displays the current state of the menu bar and menus at the top of the dialog box.

Edit»Panel

The **Panel** command opens the Edit Panel dialog box, where you can set the following options:

The **Source Code Connection** section of the Edit Panel dialog box has the following options:

- **Constant Name**—Sets the resource ID for the panel. Pass this resource ID to `LoadPanel` to load the panel into memory. The constant name is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a constant name, the User Interface Editor assigns a constant name when you save the `.uir` file.

- **Callback Function**—Specifies the name of the function to call when an event is generated on the panel. Naming a callback function is optional.

The **Panel Settings** section of the Edit Panel dialog box has the following options:

- **Panel Title**—Sets the title of the panel.

- **Menu Bar**—Sets the name of the menu bar.

- **Close Control**—Designates which control on your panel causes the panel to close.

- **Top**—Sets the barrier for the top edge of the panel, in pixels.

- **Left**—Sets the barrier for the left edge of the panel, in pixels.

- **Height**—Sets the barrier for the height of the panel, in pixels.

- **Width**—Sets the barrier for the width of the panel, in pixels.

- **Scroll Bars**—Enables or disables scroll bars on the panel.

- **Auto-Center Vertically (when loaded)**—Automatically centers the panel in the vertical center of your monitor.

- **Auto-Center Horizontally (when loaded)**—Automatically centers the panel in the horizontal center of your monitor.

- **Other Attributes**—Sets behavior and feature attributes of the panel.

From the Quick Edit Window, you can perform high-level edits on the panel. The mode tools operate like the tools in the main User Interface Editor.

The **Attributes for Child Panels** section of the Edit Panel dialog box has the following options:

- **Frame Style**—Sets the styles of the frame in the child panel.
- **Frame Thickness**—Sets the frame thickness in the child panel.
- **Title Bar Style**—Sets the title bar style in the child panel.
- **Title Bar Thickness**—Sets the title bar thickness in the child panel.
- **Size Title Bar Height to Font**—Sets the size of the title bar to match the height of the font in the title bar.
- **Title Style**—Sets the style of the title in the child panel.

Edit»Control

The **Control** command opens a dialog box in which you can interactively set many attributes of a control. You also can double-click a control to open this dialog box. The dialog box can have various sections including **Source Code Connection**, **Control Settings**, **Control Appearance**, **Quick Edit Window**, and **Label Appearance**. The sections available in the dialog box for a selected control vary slightly depending on the type of control that you are editing.

The **Source Code Connection** section of the edit control dialog box contains the following options:

- **Constant Name**—The User Interface Editor appends the constant name to the panel resource ID to form the ID for the control. The ID identifies the control in any control-specific functions, such as `GetCtrlVal` and `SetCtrlAttribute`. The ID is defined in the `.h` file that LabWindows/CVI generates when you save the `.uir` file. If you do not assign a constant name, the User Interface Editor assigns one for you when you save the `.uir` file.
- **Callback Function**—Type the name of the function to call when an event is generated on the control. Naming a callback function is optional.

The **Control Settings** section of the Edit Control dialog box displays specific attributes for the type of control that you are editing. This section contains the data-specific attributes for the control.

Ring controls and list boxes have a **Label/Value Pairs** button in the **Control Settings** section of the Edit Control dialog box.

The **Control Appearance** section of the Edit Control dialog box displays specific attributes for the type of control that you are editing. The section contains attributes pertaining to the physical appearance of the control.

From the Quick Edit Window, you can perform high-level edits on the control. The mode tools operate like the tools in the main User Interface Editor. The Quick Edit Window also immediately reflects any changes you make in other sections of the dialog box.

The **Label Appearance** section of the Edit Control dialog box contains attributes pertaining to the physical appearance of the control label.

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

Edit Label/Value Pairs

Use the Edit Label/Value Pairs dialog box to create and edit the contents of ring and list box controls.

The following items appear in this dialog box:

- **Data Type**—Selects the data type of the values in the control.
- **Precision**—Selects how many digits the control displays to the right of the decimal point.
- **Label**—Specifies a label that appears on the ring or slide control. To add a label to a ring or list box control list, type the label in the text box and press <Enter>. The highlight moves to **Value**.
- **Value**—Specifies the value, constant name, or expression associated with the label entered in **Label**. To add a value to a ring or list box control list, type the value in the text box. You can use a constant name or any other valid C expression. Press <Enter> to add the label and value to the ring or list box control list.
- **Below**—Inserts a blank line below the selected line in the list box.
- **Above**—Inserts a blank line above the selected line in the list box.
- The list box below the **Label** and **Value** text boxes displays the labels and the values of items that appear on the ring or list box control.
- **Cut**—Removes the selected line from the list and places it in the clipboard.
- **Copy**—Copies the selected line to the clipboard.
- **Paste**—Inserts the label and value line contained in the clipboard below the selected line in the list box.
- **OK**—Accepts the entries in the list box, then removes the dialog box.
- **Cancel**—Cancels changes and removes the current dialog box from the screen.
- **Help**—Displays the help topic for the Edit Label/Value Pairs dialog box.

Edit»Tab Order

Each control on a panel has a position in the tab order. The tab order determines which control becomes the next active control when the user presses <Tab> or <Shift-Tab>.

When you create a control, it positions itself at the end of the tab order. When you copy and paste a control, the tab position of the pasted control is immediately before the control you copied. Select **Edit»Tab Order** to display the Edit Tabbing Order dialog box.

Click a control with the pointer cursor to change the tab position of a control to the number in the **Click to set to** box.

You can change the cursor to the eyedropper cursor by holding down the <Ctrl> key. Click a control with the eyedropper cursor to change the number in the **Click To Set To** box to the current tab position associated with the control.

Click the **OK** button to accept the new tab order.

Click the close button to erase the new tab order and restore the original tab order. For each control, the original tab order appears in dim display to the right of the new tab order you enter.

Edit»Set Default Font

Select **Edit»Set Default Font** to make the font of the currently selected control the default control font. If you also select the label or select only the label, the font of the label becomes the default label font. Newly created controls inherit the default fonts.

Edit»Apply Default Font

Select **Edit»Apply Default Font** to set the font of the currently selected control and/or label to the default control font and/or default label font.

Edit»Control Style

Use the **Control Style** command to change the style of the selected control. For example, you can change a ring slide control to a ring knob control, and the label/value pairs remain intact.

Edit»Edit/Create Custom Controls

You can create custom controls to save control configurations. LabWindows/CVI saves your list of custom controls between sessions. To use a custom control, select **Create»Custom Controls** and select the control you want to use. You can edit the saved configurations of the custom controls by double-clicking the control name and opening the Edit Control dialog box.

To create a custom control, create a control as you normally would. Select **Create»Custom Controls»Edit** to display the Edit Custom Controls dialog box, which contains the following options:

- **Add**—Opens a dialog box that contains a list of all the controls on the user interface panel, a preview of the control appearance, and the name that will appear in the **Create** menu. Select the control you want to add as a custom control and click **OK**.

- **Delete**—Removes the selected control from the **Create** menu.

- **Edit**—Opens the Edit Control dialog box.

- **Edit Name**—Opens a dialog box in which you can change the name of the control.

- **Code**—Opens a dialog box in which you can associate the following files with the control:

  - **Program File**—A `.fp` file, source file, or object file that implements additional functionality for the control. When you select **Code»Generate»All Code** or **Generate Custom Control Code**, LabWindows/CVI automatically adds the program file to the project.

  - **Header File**—The include file that accompanies the program. When you select **Code»Generate»All Code** or **Generate Custom Control Code**, LabWindows/CVI includes the header file at the top of the source file.

  - **Template File for main**—Contains code that LabWindows/CVI imports into the `main` function when you select **Code»Generate»All Code** or **Generate Custom Control Code**. LabWindows/CVI inserts the code following the call to `LoadPanel`.

  - **Operate in UI Editor**—In operate mode, LabWindows/CVI automatically generates a DLL, which allows the control to behave as it would in a complete application. You can interact with the control in operate mode.

- **Move Up**—Moves the selected control up one line.

- **Move Down**—Moves the selected control down one line.

- **OK**—Accepts the modifications you made in the Edit Custom Controls dialog box and closes the dialog box.

- **Cancel**—Closes the dialog box without accepting any changes.

LabWindows/CVI provides the following custom controls:

- Ok Button

- Quit Button

- Toolslib Controls—LabWindows/CVI controls configured to provide additional functionality.

  - Radio Group

  - Animation Control

  - Combo Box

- – Tab Control

- – File Browser

- – Hot Ring

- – Path Control

- – Password Control

- – 3D Graph

- – Graph Cursors

- – Horizontal Scroll Bar

- – Vertical Scroll

- – NI-DAQmx IO Controls—If you have NI-DAQmx installed, you can select from a list of commonly used NI-DAQmx controls, including a task control, a relay control, and a switch control.

View Menu for the User Interface Editor

This section explains how to use the commands in the User Interface Editor **View** menu.

View»Find UIR Objects

Use the **Find UIR Objects** command to locate objects in user interface resource (`.uir`) files. When you select this command, the Find UIR Objects dialog box opens.

Select the type or types of objects you want to search for in the left column of the dialog box.

- • **Search By**—Specifies the search criterion. The following choices are available:
   - – **Constant Prefix**—Valid for panels and menu bars
   - – **Constant Name**—Valid for controls, menus, and menu items
   - – **Prefix + Constant Name**—Valid for all
   - – **Callback Function Name**—Valid for all, except menu bars
   - – **Label**—Valid for all, except menu bars

   The search criterion you select determines the name of the second item in the Find UIR Objects dialog box. Enter the text you want to search for into this control. To view a list of all the strings in the file that match the current **Search By** criterion, click the arrow to the right of the string control or use the up and down arrow keys.

- • **Find *Item***—Specifies the types of UIR objects to search for.

- • **Wrap**—Continues the search at the beginning of the file after reaching the end of the file.

- • **Case Sensitive**—Finds only the instances of the specified text that match exactly.

- **Whole Word**—Finds the specified text only when it is surrounded by spaces, punctuation marks, or other characters not part of a word. LabWindows/CVI treats the characters A through Z, 0 through 9, and underscore (_) as parts of a word.

- **Regular Expression**—Causes LabWindows/CVI to treat certain characters in the search string control as regular expression characters instead of literal characters. Refer to the table in the *Regular Expression Characters* section of Chapter 4, *Source and Interactive Execution Windows*, for a list.

- **Find**—Performs the search. If any user interface objects match, a different Find UIR Objects dialog box appears.

  You can browse through the list of matches in the new Find UIR Objects dialog box. As you come to each object, its callback function name and constant name appear, and LabWindows/CVI highlights the object in the .uir file. The Find UIR Objects dialog box contains the following buttons:

  – **Find Prev**—Searches backward for the previous matching object.

  – **Find Next**—Searches forward for the next matching object.

  – **Edit**—Terminates the search and opens the Edit dialog box for the user interface object currently highlighted.

  – **Stop**—Terminates the search.

View»Show/Hide Panels

The **Show/Hide Panels** command has a submenu.

Use this submenu to show all panels, hide all panels, or select individual panels you want to view in the User Interface Editor.

View»Bring Panel to Front

The **Bring Panel to Front** command has a submenu that lists all panels. Select a panel to bring to the front for editing.

View»Next Panel

The **Next Panel** command brings the next panel in the current .uir file to the front for viewing and editing.

View»Previous Panel

The **Previous Panel** command brings the previous panel in the current .uir file to the front for viewing and editing.

View»Preview User Interface Header File

The **Preview User Interface Header File** command opens a new Source window with a preview of the header file that LabWindows/CVI generates when you save the .uir file in the User Interface Editor.

Create Menu for the User Interface Editor

Use the **Create** menu to create panels, menu bars, and controls. For more information about each type of control you can create, refer to the *LabWindows/CVI Help*.

Arrange Menu for the User Interface Editor

This section explains how to use commands in the **Arrange** menu of the User Interface Editor.

Arrange»Alignment

Use the **Alignment** command to align controls on a panel. You can use the mouse to select a group of controls by dragging over them or <Shift-Click> on each item you want to include in the group. Then you can select an alignment method from the submenu. The options on the **Alignment** command submenu are as follows:

Left Edges vertically aligns the left edges of the selected controls to the left-most control.

Horizontal Centers vertically aligns the selected controls through their horizontal centers.

Right Edges vertically aligns the right edges of the selected controls to the right-most control.

Top Edges horizontally aligns the top edges of the selected controls to the upper-most control.

Vertical Centers horizontally aligns the selected controls through their vertical centers.

Bottom Edges horizontally aligns the bottom edges of the selected controls to the lower-most control.

Arrange»Align

The **Align** command performs the same action as the **Alignment** command, using the option you last selected in the **Alignment** command submenu.

Arrange»Distribution

Use the **Distribution** command to distribute controls on a panel. Select a group of controls by dragging the mouse over them or pressing <Shift-Click> on each item you want to include in the group. Then you can select a distribution method from the submenu. The options on the **Distribution** command submenu are as follows:

Top Edges sets equal vertical spacing between the top edges of the controls. The upper-most and lower-most controls serve as anchor points.

Vertical Centers sets equal vertical spacing between the centers of the controls. The upper-most and lower-most controls serve as anchor points.

Bottom Edges sets equal vertical spacing between the bottom edges of the controls. The upper-most and lower-most controls serve as anchor points.

Vertical Gap sets equal vertical gap spacing between the controls. The upper-most and lower-most controls serve as anchor points.

Vertical Compress compresses the spacing of controls to remove any vertical gap between the controls.

Left Edges sets equal horizontal spacing between the left edges of the controls. The left-most and right-most controls serve as anchor points.

Horizontal Centers sets equal horizontal spacing between the centers of the controls. The left-most and right-most controls serve as anchor points.

Right Edges sets equal horizontal spacing between the right edges of the controls. The left-most and right-most controls serve as anchor points.

Horizontal Gap sets equal horizontal gap spacing between the controls. The left-most and right-most controls serve as anchor points.

Horizontal Compress compresses spacing of the controls to remove any horizontal gap between the controls.

Arrange»Distribute

The **Distribute** command performs the same action as the **Distribution** command, using the option you last selected in the **Distribution** command submenu.

Arrange»Control ZPlane Order

The **Control ZPlane Order** option lets you set the sequence in which overlapped controls are drawn. Controls are always drawn in order, from the back to the front of the z-plane order. The **Control ZPlane Order** submenu contains four commands:

- **Move to Front** moves the control to the front of the z-plane order so it is drawn last.

- **Move to Back** moves the control to the back of the z-plane order so it is drawn first.

- **Move Forward** moves the control one place forward in the z-plane order.

- **Move Backward** moves the control one place backward in the z-plane order.

Arrange»Center Label

The **Center Label** command centers the label of the selected control.

Arrange»Control Coordinates

The **Control Coordinates** command opens the Control Coordinates and Dimensions dialog box in which you can interactively set the width, height, top, and bottom of all selected controls, labels, digital displays, and legends.

Code Menu for the User Interface Editor

This section explains how to use the commands in the User Interface Editor **Code** menu. Use the commands in the **Code** menu to use CodeBuilder to generate code automatically based on a .uir file you are creating or editing.

Code»Set Target File

Use the **Set Target File** command to specify to which file LabWindows/CVI generates code. Selecting this command opens the Set Target File dialog box. By default, LabWindows/CVI places the generated code in a new window, unless a source (.c) file is open. Then, that source file is the default target file. CodeBuilder uses the same target file as the function panel target file, except when the function panel target file is the Interactive Execution window.

To set a target file, select a file from the list in the Set Target File dialog box and then click **OK**. You can select from all open source files or a new window.

Code»Generate

Access the CodeBuilder features of LabWindows/CVI through the **Generate** menu item. The commands in the **Generate** menu produce code to the target file based on the `.uir` file. The code produced by the **Generate** menu uses the bracket styles you specify with the **Options»Bracket Styles** command in the Source window. The **Generate** command has a submenu that contains the following commands:

- **All Code**—Generates code to accompany the `.uir` file.
- **Main Function**—Generates code for the `main` function.
- **All Callbacks**—Generates code for all the callback functions.
- **Panel Callbacks**—Generates code for the callback function associated with a panel.
- **Control Callbacks**—Generates code for the callback functions associated with one or more controls.
- **Menu Callbacks**—Generates code for menus and menu items connected to callback functions.

The **All Callbacks** command is available when any of the **Panel Callback**, **Control Callbacks**, or **Menu Callbacks** commands are available.

The **Panel Callback** command is available if you specified a callback function for the currently active panel. The **Control Callbacks** command is available if you specified callback functions for any of the currently selected controls. The **Menu Callbacks** command is available if you have a menu bar that contains items for which you specified a callback.

When you generate code to accompany a `.uir` file, LabWindows/CVI places the skeleton code in the target file. You must save the `.uir` file before you can generate any code based on that file. When you save a `.uir` file, LabWindows/CVI generates a header (`.h`) file with the same name. This `.h` file and `userint.h` are included in the source file.

If you try to generate the same function more than once, the Generate Code dialog box appears. Each previously generated code fragment appears highlighted. Click the appropriate button in the Generate Code dialog box to replace the existing function, insert a new function, or skip to the next generated function.

Generating All Code

Selecting **Code»Generate»All Code** opens the Generate All Code dialog box. Use the Generate All Code dialog box to specify the following options for the generated code:

- Choose to create a new project for the generated source file or add the generated file to the currently loaded project.

- Select the panel or panels that the main function loads and displays at run time. LabWindows/CVI automatically assigns a default panel variable name for each panel in the .uir file. You can modify the panel variable name.

- Select the callback function or functions that terminate the program. For a CodeBuilder program to terminate successfully, you must include a call to QuitUserInterface.

✎ **Note** Callback functions associated with close controls are automatically checked in the **Program Termination** section of the Generate All Code dialog box. You can define a control as a close control in the Edit Panel dialog box by selecting **Edit»Panel**.

- If you have ActiveX servers in your panel, the Generate All Code dialog box lists the ActiveX servers corresponding to all the controls that you created in the User Interface Editor window. For each server that you select, LabWindows/CVI runs the ActiveX Controller Wizard when you generate all code.

When you choose **Code»Generate»All Code**, LabWindows/CVI produces the #include statements, the variable declarations, the function skeletons, and the main function and places them in the target file. The callback functions you selected to terminate program execution include a call to the User Interface Library QuitUserInterface function.

Unless you selected the **Code»Preferences»Always Append Code to End** option, LabWindows/CVI places the skeleton code for each callback function at the cursor position in the target file. If the cursor is inside an existing function, LabWindows/CVI repositions the cursor at the end of that function before inserting the new function. CodeBuilder places all functions of one type (panel callback, control callback, or menu callback) together in the source file. Any panel callbacks are placed first in the source file, control callbacks are placed next, and menu callbacks are placed last. You can set the default settings of generated code by selecting **Code»Preferences**.

Function skeletons for control and panel callbacks include the complete prototype, the proper syntax, a return value, and a switch construct containing a case for each default control or panel event. Function skeletons for menu callbacks include the complete prototype and open and close brackets. You can set the default events by selecting **Code»Preferences**. You can set the location of the open and close brackets by selecting **Options»Bracket Style** in a Source window.

Generating the main Function

Note   If you previously selected the **Code»Generate»All Code** command, you do not have to execute this command. Use this command only when you want to replace the main function to add or change the panels to be loaded at run time.

Selecting **Code»Generate»Main Function** opens the Generate Main Function dialog box. Use this dialog box to select the following options:

• Choose the panels that the main function loads and displays at run time. LabWindows/CVI automatically assigns a default panel variable name for each panel in the .uir file.

• Select **Generate WinMain( ) Instead of main( )** to use WinMain instead of main for your main program. In LabWindows/CVI, you can use either function as your program entry point. When linking your application in an external compiler, it is easier to use WinMain.

When you choose **Code»Generate»Main Function**, LabWindows/CVI produces the #include statements, the variable declarations, and the main function, and places them in the target file.

Note   If the source file contains only the main function and the #include statements and you have not yet created the appropriate callback functions, you might get an error when trying to run the project. When the main function calls LoadPanel, LabWindows/CVI generates a non-fatal error for each callback function it cannot find.

If your project target is a DLL, neither WinMain nor main are generated. Instead, CodeBuilder generates a DLLMain function and places the bulk of the User Interface function calls in a function called InitUIForDLL. Call InitUIForDLL in your DLL at the point you want to load and display panels.

When you link your executable or DLL in an external compiler, you must include a call to InitCVIRTE in WinMain, main, or DLLMain (or DLLEntryPoint for Borland C/C++). In a DLL, you also must include a call to CloseCVIRTE. CodeBuilder automatically generates the necessary calls to InitCVIRTE and CloseCVIRTE in your WinMain, main, or DLLMain function and also automatically generates a #include statement for the cvirte.h file.

Generating All Callbacks

When you select **Code»Generate»All Callbacks**, LabWindows/CVI produces the #include statements and the callback function skeletons and places them in the target file.

Generating Panel Callbacks

Before you can choose **Code»Generate»Panel Callback**, you must select a panel in the
`.uir` file.

When you select **Code»Generate»Panel Callback**, LabWindows/CVI produces the
`#include` statements and the function skeleton for the active panel and places them in the
target file.

Generating Control Callbacks

Before you can choose **Generate»Control Callbacks**, you must select at least one control.

When you select **Code»Generate»Control Callbacks**, LabWindows/CVI produces the
`#include` statements and the function skeleton for each selected control and places them in
the target file.

You also can generate a control callback function skeleton by right-clicking a control in the
`.uir` file and selecting **Generate Control Callback** from the context menu.

Generating Menu Callbacks

Selecting **Code»Generate»Menu Callbacks** opens the Select Menu Bar Objects dialog box.
Select the menu bar objects for which you want to generate callbacks and then click **OK**.

When you click **OK**, LabWindows/CVI produces the `#include` statements, the function
prototypes, and the opening and closing brackets for each callback function.
LabWindows/CVI does not generate switch construct or case statements because the usual
default events do not apply to menu callback functions. You must add the code to implement
the actions you want to take place when a user selects a menu bar item.

Code»View

Use the **Code»View** command to look at code for a given callback function.

To view the code for a function from the `.uir` file, select a panel or control and then select
View»Panel Callback or **View»Control Callback**. The source file that contains the callback
function appears with the function name highlighted. You also can view the code for a control
callback function by right-clicking the control and selecting **View Control Callback** from the
context menu.

When you choose the **View** command for a callback function, LabWindows/CVI searches for
that function in all open files and in all files in the project.

The **View** command is useful because the callback functions for one user interface can be in
several different files, and scrolling the source code is not efficient. With the **View** command,

you can move instantly from the user interface file to an object callback function, whether the source file is open or closed.

When you finish reviewing the code, you can return instantly to the .uir file from the source file. To return to the .uir file, place the cursor on the callback function name or constant name of the user interface object you want to go to and select **View»Find UI Object** in the Source window.

✎   **Note**   You cannot use the **View** command for menu callback functions.

Code»Preferences

Use the **Preferences** command to change the default settings of case statements generated for control callback functions and panel callback functions or to specify the target file location for generated code.

Use the **Default Panel Events** or **Default Control Events** commands to select which events LabWindows/CVI places into the switch construct of the code for panel or control callback functions, respectively. You can choose from several events, and you can choose to add a default switch case. Selecting **Code»Preferences»Default Panel Events** opens the Panel Callback Events dialog box. Selecting **Code»Preferences»Default Control Events** opens the Control Callback Events dialog box.

To set the **Default Panel Events** or **Default Control Events**, select the events you want to include in the code as case statements and then click **OK**. For each option you choose, LabWindows/CVI includes in the source code a case statement that corresponds to this option.

✎   **Note**   Default control events are ignored for timer control callbacks, for which the only event cases are EVENT_TIMER_TICK and EVENT_DISCARD.

When you select **Always Append Code to End**, LabWindows/CVI places the skeleton code for each callback function at the end of the target file. If you do not select this option, LabWindows/CVI places newly generated code at the current position of the cursor in the target file.

Run Menu for the User Interface Editor

The commands that appear in the User Interface Editor **Run** menu are the same as the commands for the **Run** menu in the Workspace window. Refer to the *Run Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information about these commands.

Library Menu for the User Interface Editor

The **Library** menu for the User Interface Editor works in the same way as the **Library** menu for the Workspace window. For more information about these commands, refer to the *Library Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Tools Menu for the User Interface Editor

The **Tools** menu for the User Interface Editor works in the same way as the **Tools** menu for the Workspace window. For more information about these commands, refer to the *Tools Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Window Menu for the User Interface Editor

The **Window** menu for the User Interface Editor works in the same way as the **Window** menu for the Workspace window. For more information about these commands, refer to the *Window Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Release/Confine Window

Use the **Release Window** command to remove the window from the Window Confinement Region.

The **Confine Window** command appears when the window is released from the workspace area. Use this command to confine the window.

Options Menu for the User Interface Editor

This section explains how to use the commands in the User Interface Editor **Options** menu.

Options»Operate Visible Panels

Use the **Operate Visible Panels** command to operate the visible panels as you would in an application program. This command has the same effect as clicking the operating tool, shown at left. When you finish operating the panel, select **Operate Visible Panels** again to return to edit mode.

Options»Next Tool

The **Next Tool** command cycles the User Interface Editor through its four modes, as described in the *User Interface Editor Overview* section.

Options»Preferences

Selecting **Options»Preferences** opens the Editor Preferences dialog box.

Use the **User Interface Editor Preferences** section to set options that affect the operation of the User Interface Editor.

- **Initial editor background color**—Determines the initial background color of the User Interface Editor.

- **Default**—Restores the initial background color of the User Interface Editor to the default color.

- **Use localized decimal symbol**—Replaces the traditional decimal symbol used by LabWindows/CVI in numeric, graph, and table controls with the decimal symbol specified in the Regional Settings Properties configuration of your Windows Control Panel.

- **Show Editor Grid Lines**—Displays a grid on user interface panels. You can use the grid lines to align and resize controls. This option is enabled by default.

- **Grid Line Type**—Specifies the appearance of the grid lines. You can select crosses, solid lines, dots, and dotted lines.

- **Grid Line Spacing**—Specifies the amount of space that appears between grid lines. The default is 25.

- **Snap to Grid Lines**—Snaps a control to the nearest grid line as you move the control on the panel or resize the control. Snapping to grid lines works only when you move or resize the control within a certain distance of a grid line.

Use the **Preferences for New Panels** section to set initial attribute values for each panel that you create in the User Interface Editor.

- **Resolution Adjustment(%)**—Specifies the degree to which LabWindows/CVI scales panels and their contents when you display them on screens with resolutions different than the one on which you create them. This option also appears in the Other Attributes dialog box that you can open from the Edit Panel dialog box.

  To programmatically override this setting, call `SetSystemAttribute` with the `ATTR_RESOLUTION_ADJUSTMENT` attribute before calling `LoadPanel` or `LoadPanelEx`.

- **Conform to System Colors**—Forces panels and the controls they contain to use the system colors. This option also appears in the Other Attributes dialog box that you can open from the Edit Panel dialog box. To programmatically set this option, call

`SetPanelAttribute` with the `ATTR_CONFORM_TO_SYSTEM` attribute. When you enable this option, you cannot change any panel or control colors.

- **Use system colors as defaults for panels and controls**—Forces LabWindows/CVI to use the system colors as the initial colors for panels and controls you create. You can change the colors without restriction.

- **Background color**, **Frame color**, **Title bar color**—You must disable two options, **Conform to System Colors** and **Use system colors as defaults for panels and controls**, to set the following options: background color, child panel frame color, and child panel title bar color. To change each of these three options in the User Interface Editor, you can use the Paintbrush tool on the background, frame, or title bar of a panel. To set these colors programmatically, use `SetPanelAttribute` with the `ATTR_BACKCOLOR`, `ATTR_FRAME_COLOR`, and `ATTR_TITLE_BACKCOLOR` attributes.

Use the **Preferences for New Controls** section of the Editor Preferences dialog box to set initial attribute values for each control that you create in the User Interface Editor.

- **Control Text Style**—Sets the initial font and text style for all new controls.

- **Label Text Style**—Sets the initial font and text style for all labels.

Click **More** to open the Other User Interface Editor Preferences dialog box.

Other User Interface Editor Preferences

- Use the **Undo Preferences** section of the Other User Interface Editor Preferences dialog box to set the number of actions you can undo for each file.

  If you want the undo buffer to empty every time you save a file, select **Purge undo actions when saving file**.

- Use the **Constant Name Assignment** section to set preferences for constant name assignment when you do not assign constant names yourself. Constant names link user GUI objects and your program. The User Interface Editor writes all assigned constant names to a header file corresponding to the `.uir` file.

  Select **Immediately assign constant names for new objects** to generate a constant name for each object as you create it. For panels and controls, the generated constant name appears in the Edit dialog box the first time you open it. For menu bars, LabWindows/CVI assigns constant names only when you exit the Menu Bar Editor. In all cases, you can freely modify the generated constant names.

  NI recommends that you leave the **Immediately assign constant names for new objects** option selected. This option makes it easier for you to use the other LabWindows/CVI features that have been designed to help you write your program to operate your user interface.

  Notice that when you enable **Immediately assign constant names for new objects**, **Assign constant names from user-defined control labels when possible** has no effect.

That is because you do not have a chance to customize the control labels before the User Interface Editor generates the constant name. Consequently, the User Interface Editor bases the constant name on the control type.

If you choose to disable the **Immediately assign constant names for new objects** option, NI recommends that you enable the **Fill in missing constant names when saving** option.

Options»Assign Missing Constants

The **Assign Missing Constants** command assigns constant names to all of the objects in the User Interface Editor that currently do not have constant names. A confirmation dialog box appears showing the number of items that do not have constant names.

Options»Save in Text Format

The **Save in Text Format** command saves the contents of the User Interface Editor in an ASCII text format. A dialog box appears prompting you to enter the pathname under which to save the text file. The extension `.tui` is recommended for such files. Do not use the `.uir` extension.

The ASCII text file contains descriptions of all the objects in the User Interface Editor. You can call `LoadPanel` and `LoadPanelEx` on `.tui` files.

Note   If you have a large number of objects in your User Interface Editor, loading a `.tui` file can take significantly longer than loading a comparable `.uir` file.

Note   The `.tui` file format in LabWindows/CVI 5.0 and later differs from previous versions. If you use `.tui` files to find differences between versions of your `.uir` files and you created `.tui` files in previous versions of LabWindows/CVI, create new baseline `.tui` files for your `.uir` files.

Options»Load from Text Format

The **Load from Text Format** command loads into a new User Interface Editor the objects defined in a file saved using the **Save in Text Format** command. A dialog box appears prompting you for the pathname of the file.

Help Menu for the User Interface Editor

The **Help** menu for the User Interface Editor works in the same way as the **Help** menu for the Workspace window. For more information about these commands, refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Help»Control Help

When you select a control, the **Control Help** command displays online help for the selected control.

4

Source and Interactive Execution Windows

Source Windows

Source windows display source code for the programs you develop. These windows behave like standard text editors. You can insert text into Source windows in the following ways:

- Type text directly into a Source window.
- Load text from an ASCII file.
- Insert code from function panels directly into Source windows.
- Drag and drop text into the Source window from the current Source window, another Source window, or other applications.

You also can drag text from the Source window into different LabWindows/CVI environment windows.

- Drag code into a Watch window to add a watch expression.
- Drag variables into the Graphical Array View, Array Display, Memory Display, and String Display windows.
- Drag code into the Interactive Execution window.

You can save a program from a Source window as an ASCII file. Source windows can contain up to one million lines with up to 1,020 characters on each line. A tab is one character for the purpose of line length limitation.

When you run or build a program in a Source window, the program must be complete and obey the syntax rules of ANSI C.

Notification of External Modification

If you have externally modified a file since you last loaded or saved it in LabWindows/CVI and the file is in a Source window, a dialog box appears when you switch back to LabWindows/CVI from another Windows application. In the dialog box, you can update the Source window from the file on disk, overwrite the file on disk with the contents of the Source window, or do nothing.

Context Menus in Source Windows

To open a context menu, right-click anywhere in the Source window. The context menu contains a set of the most commonly used menu commands from the Source window menu bar, plus additional options not available through menus. The set of commands is different depending on whether you right-click over the text editing area or over the line number or line icon area. The following options are not available through menus.

- **Cut Line**—Removes the entire line and places it on the clipboard.

- **Copy Line**—Copies the line to the clipboard. The line remains in the source file.

- **Generate DAQ Example Code**—This option generates code that shows how to run the selected task in a program.

  Selecting this menu item opens the Generate DAQ Example Code dialog box. You must specify the Run Task function name and the target source file. When you click **OK**, LabWindows/CVI generates example code that shows how to run the selected task in a program. The generated code includes source and header files that define the Run Task function. LabWindows/CVI automatically adds these files to the project. The current source file is modified to call the run task function.

  The generated code has no link to the original task. Any modifications you make to the original task will not be automatically reflected in the generated code.

  This menu item is enabled only when you place the cursor on a call to `DAQmxLoadTask` or the name of the task creation function of a project-based task.

- **Copy DAQ Task to Project**—This option creates a new project-based task that is a copy of the selected MAX-based task.

  Selecting this menu item opens the Copy DAQ Task to Project dialog box. You must specify a task name, task creation function name, and target directory. When you click **OK**, LabWindows/CVI completes the following actions:

  – Generates source (`.c`), header (`.h`), and binary (`.mxb`) files and adds them to the project.

  – Replaces the call to `DAQmxLoadTask` with a call to the new task creation function.

  – Adds an include statement for the generated header file.

  The generated source and header files define the task creation function. This function contains the code necessary to create a task that is equivalent to the selected MAX-based task. The generated binary (`.mxb`) file contains a binary description of the task that is used when you edit the task in the DAQ Assistant.

  You might want to copy a MAX-based task to a project-based task to facilitate sharing the task definition among multiple developers or storing the task definition in a source code control system.

The generated files have no link to the original MAX-based task. Any modifications you make to the original MAX-based task will not be reflected in the binary (`.mxb`) file or the generated code.

This menu item is enabled only when you place the cursor on a call to `DAQmxLoadTask`.

• **Show DAQ Configuration Code**—This option opens an untitled Source window containing the source code required to create a task that is equivalent to the selected MAX-based task. The source code that appears is the same source code that would be generated by the **Copy DAQ Task to Project** menu item.

This menu item is enabled only when you place the cursor on a call to `DAQmxLoadTask`.

Interactive Execution Window

You can execute selected portions of code in the Interactive Execution window. Unlike the Source window, you do not need a complete program in the Interactive Execution window. For instance, you can execute C variable declarations and assignment statements without declaring a `main` function.

Use the Interactive Execution window to test portions of code before you include them in your main program. Also, you can use the Interactive Execution window to execute functions exported by a loaded instrument or by a file in the project if the project has been linked. The Interactive Execution window can access functions and data declared as global in a Source window, but a Source window has no access to the functions and data declared in the Interactive Execution window.

When you execute a function from a function panel, LabWindows/CVI inserts the function call into the Interactive Execution window for execution. In this way, the Interactive Execution window keeps a record of the functions you execute from function panels.

When LabWindows/CVI copies a function call from a function panel to the Interactive Execution window for execution, it inserts the code after all the existing lines. LabWindows/CVI also inserts an include statement for the header file associated with the function in the Interactive Execution window if you have not already included it. When you execute a function call from a function panel, LabWindows/CVI automatically excludes all previous lines in the Interactive Execution window. An excluded line is dimmed and the LabWindows/CVI compiler ignores it.

Because LabWindows/CVI automatically excludes all declarations when you execute code in the Interactive Execution window, you must avoid placing executable statements on the same line as declarations in the Interactive Execution window. Auto-exclusion also occurs when you type a line of code beneath a line that has just been executed. You can manually exclude and include lines with the **Edit»Toggle Exclusion** command.

Declarations in the Interactive Execution window remain in effect until you select
Build»Clear Interactive Declarations or **Edit»Clear Window**.

Use the following rules for executing code in the Interactive Execution window.

- Make sure that data declarations precede any program statements. Function declarations
  also are necessary unless you disable the **Require function prototypes** option in the
  Build Options dialog box in the Workspace window.

- You cannot include function definitions in the Interactive Execution window.
  LabWindows/CVI treats the following statements the same:

  ```
  extern int fn (void);
  int fn (void);
  ```

  LabWindows/CVI treats the following statements as errors:

  ```
  static int fn (void);
  static int fn () {}
  ```

- LabWindows/CVI treats all global data declarations in the Interactive Execution window
  as if they are declared as static unless the `extern` keyword precedes them. If the `extern`
  keyword precedes them, the global declaration must exist in a loaded instrument or in a
  file in the project.

  The following data declaration is invalid in the Interactive Execution window:

  ```
  extern int x=6;
  ```

You can drag text from the Interactive Execution window into different LabWindows/CVI
environment windows.

- Drag code into a Watch window to add a watch expression.
- Drag variables into the Graphical Array View, Array Display, Memory Display, and
  String Display windows.

Using Subwindows

The Source and Interactive Execution windows support subwindows so that you can have two
scrollable editing areas for the same file. To create a subwindow from any of these windows,
use the mouse to drag the thin line beneath the menu bar (or toolbar if you selected
View»Toolbar) to a lower position in the window. You then can switch between the
subwindows by pressing <F6> (if you have default shortcut keys enabled) or by clicking in a
subwindow.

Selecting Text in the Source and Interactive Execution Windows

Certain LabWindows/CVI commands require that you select the block of text to which the next command applies. In LabWindows/CVI, you can select a range of characters, a range of lines, or a range of columns. When you select a block of text, it is highlighted on the screen.

To select text with the keyboard, press <Shift> as you move the keyboard cursor over the text you want to select. You can use the <Shift> key in combination with any of the keyboard commands for moving the keyboard cursor or scrolling the window.

To select text with the mouse, click the first character you want to select and drag the mouse over the remaining characters. To select a word, double-click the word. To select a line, triple-click the line. If you make a mistake while selecting text, click the mouse or press <Esc> to cancel the selection.

LabWindows/CVI provides three modes for selecting text depending on the state of the graphical icon at the bottom of the window, as illustrated in the following figures.

Character Select Mode highlights all characters from where you begin selecting text to where you end the selection.

Line Select Mode highlights full lines of text.
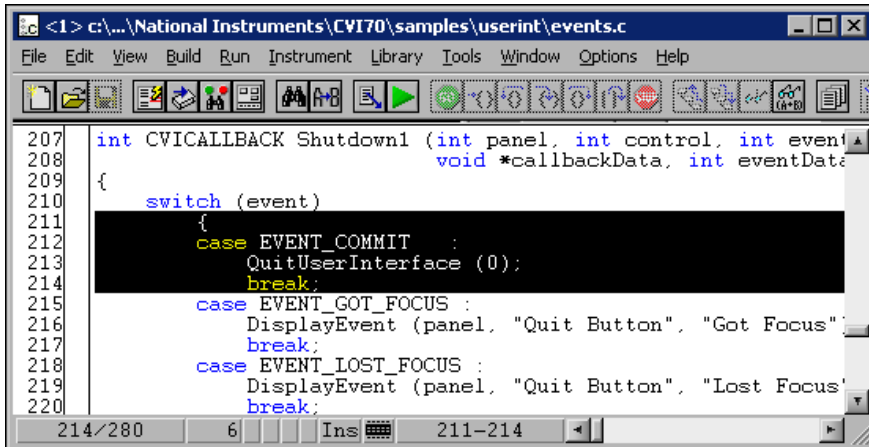
Column Select Mode highlights a rectangular block of text.
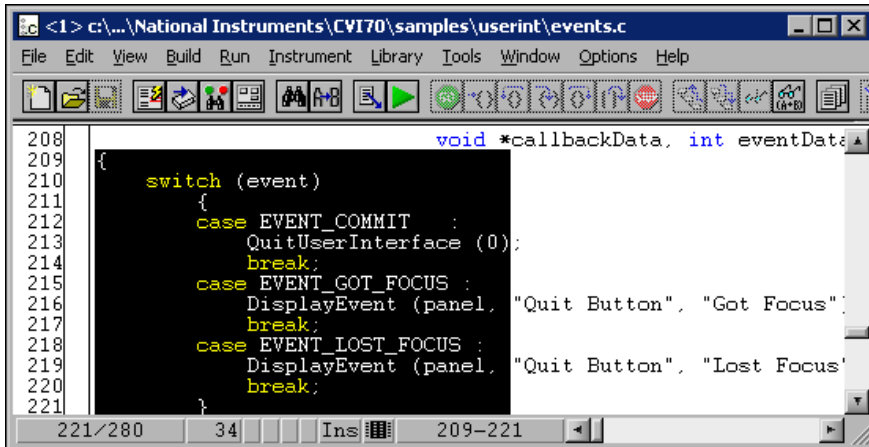
You can cycle through these three modes by pressing <Ctrl-Insert> or by clicking the graphical icon at the bottom of the window.

File Menu for the Source and Interactive Execution Windows

This section explains how to use the commands in a Source and Interactive Execution window **File** menu.

File»New, Open, Save, Save All, Most Recently Closed Files, and Exit LabWindows/CVI

The **New**, **Open**, **Save**, **Save All**, **Most Recently Closed Files**, and **Exit LabWindows/CVI** commands in the **File** menu of the Source window work like the commands in the **File** menu of the Workspace window. For more information about these commands, refer to the *File Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

File»Open Quoted Text

The **Open Quoted Text** command opens `.c`, `.h`, `.fp`, and `.uir` files that appear by name in the active window. If you select **Open Quoted Text** when the text cursor in the active window is on a line that contains a filename in quotation marks or angle brackets, that file opens in the corresponding window type.

File»Save As, Save Copy As, Add File to Project, and Read Only

These commands work like the commands in the **File** menu of the User Interface Editor. For more information about these commands, refer to the *File Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*.

File»Close

The **Close** command closes the active window. If you have modified the contents of the window since the last save, LabWindows/CVI prompts you to save the file to disk.

✎ **Note**   The **Hide** command replaces the **Close** command in the Interactive Execution window. The **Hide** command visually closes the Interactive Execution window but retains the contents in memory.

File»Print

The **Print** command prints the window contents to a printer or a file.

Edit Menu for the Source and Interactive Execution Windows

This section explains how to use the commands in a Source or Interactive Execution window **Edit** menu. Use the commands in the **Edit** menu to edit text in Source windows and the Interactive Execution window.

Edit»Undo and Redo

✎   **Note**   LabWindows/CVI disables **Undo** and **Redo** until you make an edit. LabWindows/CVI disables the **Cut** and **Copy** commands until you select text and disables **Paste** until you place text on the clipboard.

The **Undo** command reverses your last edit action. LabWindows/CVI stores editing actions in a stack so that sequential **Undo** commands reverse a history of your edit actions. You can set the size of this stack using the **Options»Editor Preferences** command. The maximum capacity of this stack is 1,000 operations.

The **Redo** command reverses your last **Undo** command. **Redo** is helpful when you use the **Undo** command to reverse a series of your edit actions and accidentally go too far. LabWindows/CVI enables the **Redo** command only when the previous action was the **Undo** command. Any other action, even moving the cursor, disables the **Redo** command.

Edit»Cut and Copy

To cut or copy text to the clipboard, select the text you want to place on the clipboard and then select **Cut** or **Copy** from the **Edit** menu. LabWindows/CVI places the selected text on the clipboard. If you used the **Copy** command, the text remains in the window. Use the **Cut** command to delete text from the window. Text you cut or copy does not accumulate on the clipboard. Every time you cut or copy text, it replaces the previous contents of the clipboard.

Edit»Paste

The **Paste** command inserts text from the clipboard.

- If you paste in character-select mode, the characters appear at the cursor on the current line.
- If you paste in line-select mode, the new lines appear above the current line.
- If you paste in column-select mode, the new block of characters appears at the cursor on the current line.
- If you select text before you paste, the contents of the clipboard replace the selected text.

You can paste the same information from the clipboard as many times as you like. Text remains on the clipboard until you use **Cut** or **Copy** again or until another application overwrites the clipboard. The **New** and **Open** commands do not erase the clipboard.

To insert text from the clipboard, move the cursor to the place you want the text inserted and select **Edit»Paste**.

Edit»Delete

The **Delete** command deletes highlighted text without placing the text on the clipboard.

Edit»Select All

The **Select All** command selects all the text in the Source window and positions the keyboard cursor at the end of the file.

Edit»Clear Window

Use the **Clear Window** command in the Interactive Execution window to clear the contents of the window. The **Clear Window** command also clears any variables declared in the Interactive Execution window.

Note The **Clear Window** command is disabled in Source windows.

Edit»Toggle Exclusion

You can specify portions of code to exclude during compilation and execution. LabWindows/CVI ignores excluded code and displays it in a different color than included code.

The **Toggle Exclusion** command marks lines in Source windows and the Interactive Execution window as excluded or included code. This command acts on single and multiple line selections.

When you work in the Interactive Execution window, lines are excluded automatically. Select **Edit»Toggle Exclusion** if you want to include these lines.

Edit»Resolve All Excluded Lines

The **Resolve All Excluded Lines** command interactively highlights the next excluded line or set of consecutive excluded lines and allows you to reinclude, comment out, conditionally compile out, delete, or skip the code.

Edit»Insert Construct

The **Insert Construct** command has a submenu of various C programming constructs. Use this command to insert a construct into your Source window at the current keyboard cursor position.

For most of these menu items, a dialog box appears in which you can fill in portions of the construct. You can press <Enter> or click **OK** without filling in the dialog box.

When you insert the construct into your program, the keyboard cursor moves to the first location in the construct where you can enter text.

You can set the location of the curly brackets in the construct by selecting **Options»Bracket Styles**.

Edit»Balance

Use the **Balance** command to find pairs of opening and closing curly braces, brackets, and parentheses. If the cursor is within (or near) a set of any of these symbols when you select the **Balance** command, LabWindows/CVI highlights all characters between them. This command is useful when you want to find a missing opening or closing symbol and a large number of these symbols are nested inside each other.

Edit»Diff

Use the **Diff** command for comparing two source files to detect any differences.

The **Diff** command has a submenu that you can use to perform the following actions:

*   Use **Diff With** to select another open source file and compare it against the current source file.

*   After you select the two files to compare, use **Synchronize at Top** to display both files starting at the top.

*   Select **Find Next Difference** to display the next point where a difference exists in the files.

*   Highlight a section from one of the files and select **Synchronize Selections** from that window to find a matching section in the other file.

*   Use **Match Criteria** to establish the number of lines that must match to mark the end of differing sections in a file.

*   Select **Ignore White Space** to compare files while ignoring spaces, tabs, or other text control characters.

*   Use **Recompare Ignoring White Space** once a difference has been found to determine if the only difference in the selections involves white space characters.

Edit»Go to Definition

When you place the text cursor on a C identifier and select **Go to Definition**, LabWindows/CVI highlights the definition of the identifier. If the definition is not available, for example, a LabWindows/CVI library function definition, LabWindows/CVI highlights the declaration of the identifier.

Edit»Go to Next Reference

When you place the text cursor on a C identifier and select **Go to Next Reference**, LabWindows/CVI highlights the next reference of the identifier.

Edit»Go Back

Returns to the previous identifier.

Edit»Show Completions

Use the **Show Completions** option to view a list of potential matches for the function or variable you are typing. LabWindows/CVI opens a drop-down list of symbol names beginning with the letters you typed. Use the arrow keys to navigate through the list and press <Enter> to insert a symbol in your code. If only one item appears in the list, LabWindows/CVI inserts that item in your code immediately.

Show Completions also lists variables, structure members, enum members, and macros after you compile the source code. If you enable **Show function prototypes and struct/union fields while typing** in the Editor Preferences dialog box, LabWindows/CVI shows structure members when you type . or –> after a variable name.

Function names are taken from function panels and browse info for user-defined functions. For browse info to be included, you must compile the source code first. Macros are included only if you have used or referenced the macro previously in the code.

Edit»Show Prototype

Use the **Show Prototype** option to view the function prototype of the current function. If you enable **Show function prototypes and struct/union fields while typing** in the Editor Preferences dialog box, LabWindows/CVI shows the prototype of the function when you type the open parenthesis after a function name. As you type in parameters, the current parameter is highlighted.

Press <Esc> to remove the prototype from the screen.

Prototype information is taken from function panels and from browse info for user-defined functions. For browse info to be included, you must compile the source code first. Macros are included only if you have used or referenced the macros previously in the code.

Edit»Find

Use the **Find** command to locate specified text in your files. When you select the **Find** command, the Find dialog box opens.

Enter the text you want to find in **Find What**. If you select text on a single line before you open the Find dialog box, the selected text appears in **Find What**. Otherwise, the text you last searched for appears in the box. To access a history of selections for **Find What**, click the arrow to the right of **Find What** or use the up or down arrow keys on your keyboard.

- **Case Sensitive**—Finds only exact matches of the specified text. For example, if you specify CHR, the **Case Sensitive** option finds CHR but not Chr.

- **Whole Word**—Finds the specified text only when the characters that surround it are spaces, punctuation marks, or other characters not considered part of a word. LabWindows/CVI treats the characters A through Z, a through z, 0 through 9, and underscore (_) as part of a word.

- **Regular Expression**—Uses certain characters in **Find What** as regular expression characters instead of literal characters. Refer to the table in the *Regular Expression Characters* section for a list of regular expression characters.

- **Multiple Files**—Includes multiple files in the search.

  To search multiple files, specify the following options:

  - **File Types**—Enter the file extension(s) of the file types you want to search. LabWindows/CVI keeps a history of the files types you select. To access the history, click the arrow to the right of **File Types** or use the keyboard arrow keys.

  - **Directory**—Browse to or enter the directory to search. The search includes all files with the specified extension in this directory and its subfolders. LabWindows/CVI keeps a history of selected directories. To access the history, use the keyboard arrow keys.

  - **Don't Search Subfolders**—Enable to search only the top-level folder you have specified. By default, the multiple files search includes subfolders.

  - **Additional Files**—Select other files you want to include in the search. **Additional Files** lists files related to the current workspace.

    Use the buttons to the right of **Additional Files** to select or deselect files quickly.

- **Selected Text Only**—Searches only within the region of highlighted text in the Source window when the highlighted text extends beyond one line. LabWindows/CVI automatically enables this option when you open the Find dialog box after selecting multiple lines of text in the Source window.

- **Wrap**—Continues searching from the beginning of the file once the search reaches the end of the file.

- **Button Bar**—Enables a dialog box for interactive searching. **Find Prev** and **Find Next** operate the same as they do in the main Find dialog box. **Stop** terminates the search,

leaving the keyboard cursor at the current position. **Return** terminates the search, leaving the keyboard cursor at the original position where you began the search. For multiple file searches, **Next File** moves the search to the next file. Selecting the **Button Bar** option disables the **Show All Matches** option.

- **Show All Matches**—Opens the Find Results window with a list of matches. The Find Results window lists the filename and line number of the matched text. Double-clicking a match in the Find Results window opens the file and highlights the matched text. If you have default shortcut keys enabled, press <F4> to cycle through the matches.

- **Find Prev**—Searches backward for the previous match.

- **Find Next**—Searches forward for the next match.

- **Cancel**—Terminates the search and closes the Find dialog box.

Regular Expression Characters

Purpose	Character	Description	Example
Wildcard matching	`.` (period)	Match 1 character	`a.t` matches `act` and `apt` but not `abort`
Matching zero or more occurrences	`*` (asterisk)	Match 0 or more occurrences of preceding character or expression	`0*1` matches `1`, `01`, `001`, and so on. `a.*` matches `act`, `apt`, and `abort`
	`+` (plus sign)	Match 1 or more occurrences of preceding character or expression	`0+1` matches `01`, `001`, `0001,...`

Purpose	Character	Description	Example
Matching either/or	? (question mark)	Match 0 or 1 occurrences of preceding character or expression	`0?1` matches `1` and `01` but not `001`
	| (pipe)	Match either the preceding or following character or expression	`a|b` matches every occurrence of `a` or `b` `abor|ut` matches every occurrence of `abort` or `about` `{if}|{else}` matches every occurrence of `if` or `else`
Matching the beginning or ending of a line	^ (caret)	Match the beginning of a line	`^int` matches any line that begins with `int`
	$ (dollar sign)	Match the end of a line	`end$` matches any line that ends with `end`
Grouping expressions	{} (curly braces)	Group characters or expressions for searches	`{if}|{else}` matches every occurrence of `if` or `else`
Matching a set	[] (brackets)	Match any one character or range listed within the brackets	`[a-z]` matches every occurrence of lowercase letters `[abc]` matches every occurrence of `a`, `b`, or `c`
	~ (tilde)	If appears immediately after the left bracket, negate the contents of the set	`[~a-z]` matches everything except lowercase letters `[a-z~A-Z]` matches all letters and the ~ character

Purpose	Character	Description	Example
Special characters	`\t` (backslash t)	Match any tab character	`\t3` matches every occurrence of a tab character followed by a `3`
	`\x` (backslash x)	Match any character specified in hex	`\x2a` matches every occurrence of the `*` character
	`\` (backslash)	Include the subsequent regular expression character in the search	`\-\?` matches every occurrence of `-` followed by `?`

Edit»Replace

The **Replace** command provides similar options as **Find**, plus the option to replace one search string with another string.

In the Replace dialog box, enter the text you want to find in **Find What** and enter in **Replace With** the new text you want to appear. Specify the same options described in the *Edit»Find* section.

Click **Find Next** to search for the next match of the search string. The Replace button bar appears with the following options:

- **Find Next**—Skips to the next occurrence of the search string without making a change.
- **Replace**—Replaces the currently selected text and goes to the next occurrence of the search string.
- **Replace All**—Finds and replaces all occurrences of the specified text without asking for confirmation.
- **Stop**—Terminates the search, leaving the keyboard cursor at the current position.
- **Return**—Terminates the search, leaving the keyboard cursor at the position where you initiated the search.
- **Next File**—Moves the search to the next file if you enabled **Multiple Files**.

Click **Replace All** to replace in the currently open file all occurrences of the search string without asking for confirmation.

Edit»Quick Search

Use the **Quick Search** command to perform an incremental search. When you select **Edit»Quick Search** or press <Ctrl-Q>, if you have default shortcut keys enabled and begin typing, the **Quick Search** command finds matches of the letters you typed. To continue searching for the same string, press <Ctrl-Q>.

Edit»Next File

If you selected the **Multiple Files** option from either the Find or Replace dialog boxes, you can move to the next file in the search list using this command.

View Menu for the Source and Interactive Execution Windows

This section explains how to use the commands in a Source and Interactive Execution window **View** menu. Use commands in the **View** menu to display line numbers and tags on source code, step through build errors, and manipulate function panels that pertain to your editing session.

View»Line Numbers

The **Line Numbers** command displays a column in the left of the Source window in which line numbers appear. A checkmark appears next to the **Line Numbers** item in the **View** menu when you activate the line number display.

View»Line Icons

The **Line Icons** command displays a column in the left of the Source window in which line icons appear. Line icons indicate the lines that you mark for breakpoints and the lines that you tag. A checkmark appears next to the **Line Icons** item in the **View** menu when you activate the line icons display.

Note   LabWindows/CVI saves line icons in the workspace file. Editing source files outside of LabWindows/CVI, however, might invalidate the positioning of associated line icons.

View»Toolbar

Use the **Toolbar** command to toggle between viewing or not viewing the Source window toolbar. This item is not available for confined Source windows.

View»Line

The **Line** command moves the cursor to the line that you specify. When you select **Line**, a dialog box appears in which you enter the number of the line where you want to position the cursor.

If you specify a line number greater than the total number of lines in the program, the cursor moves to the last line of the program.

View»Beginning/End of Selection

The **Beginning/End of Selection** command toggles the window between the beginning and the end of a highlighted block of text. This is useful when you want to verify a selected block of text that is larger than the Source window.

View»Toggle Tag

The **Toggle Tag** command toggles the tag associated with the active line. Use tags to mark lines of code that you want to revisit quickly.

View»Next Tag

Use the **Next Tag** command to go to the next tagged line. Selecting **Next Tag** repeatedly takes you to all tagged lines in the windows you specify using the **Tag Scope** command.

View»Previous Tag

Use the **Previous Tag** command to go to the previous tagged line. Selecting **Previous Tag** repeatedly takes you to all tagged lines in the windows you specify using the **Tag Scope** command.

View»Tag Scope

Use the **Tag Scope** command to set which files you want to search with **Next Tag** and **Previous Tag**. You can set the scope to the current window, all open windows, or all files.

View»Clear Tags

The **Clear Tags** command opens a dialog box in which you can remove existing Source window tags.

View»Function Panel History

The **Function Panel History** command displays a scrollable list of the function panels you have used during the current LabWindows/CVI session. You can display function panels from the list as new windows, or you can overwrite the current Function Panel window.

View»Function Panel Tree

The **Function Panel Tree** command displays the Select Function Panel dialog box for the most recently used function panel.

View»Recall Function Panel

Use the **Recall Function Panel** command to display the function panel corresponding to the call. **Recall Function Panel** not only finds and displays the panel but also sets the panel controls so that they contain the parameter values that appear in the function call. After modifying one or more controls, you can replace the original call with the modified call.

Selecting the Recall Function Panel Command

Before you select **Recall Function Panel**, you must indicate the function panel you want to recall. One method is to place the cursor on a line that contains a function call or a portion of a function call. You can select part of a line, provided that the part contains a function call.

If a line contains multiple function calls or one function call embedded within another, you can resolve the ambiguity by placing the cursor on or immediately after the function name.

After you indicate the function call, select **View»Recall Function Panel**. The function panel for that function appears, and the controls contain the parameter values from the call.

Recalling a Function Panel from a Function Name Only

You can recall a panel from a function name without specifying any of the parameters. If you place the keyboard cursor on or immediately after a function name, **Recall Function Panel** recognizes the function name even if a parameter list does not follow it. Thus, you can type a function name into the Source window and select **Recall Function Panel**.

Also, you can use the **Find Function Panel** command to open a function panel from a function name or a portion of a function name.

Multiple Panels for One Function

If the selected function appears in more than one Function Panel window, LabWindows/CVI displays a list of panels. Select one by highlighting the panel name and pressing <Enter> or by double-clicking the panel name.

Multiple Functions in One Function Panel Window

If the selected function matches a Function Panel window that contains multiple function panels, LabWindows/CVI attempts to match the panel to function calls on the lines surrounding the selected call. After the panel appears, you can check how many lines were

matched to the Function Panel window by looking at the Source window. LabWindows/CVI highlights the matched lines.

If you select multiple lines before executing the **Recall Function Panel** command, all function calls in the selected lines must appear in one Function Panel window, and the order in which the window generates the calls must be identical to the order in which they appear in the selected lines. Otherwise, an error message appears.

Syntax Requirements for the Recall Function Panel Command

You do not need to compile the file before you invoke the **Recall Function Panel** command. In fact, the function call you select does not need to be syntactically valid. The only requirement is that you must spell and capitalize the name of the function correctly. If you do not spell and capitalize the function name correctly, LabWindows/CVI displays an error message indicating that the panel could not be found.

View»Find Function Panel

When you select **Find Function Panel**, a dialog box appears in which you can enter the name of a function. You can enter just a substring, and **Find Function Panel** finds all functions that contain that substring anywhere in their names. For instance, if you enter `ctrl` and click **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use regular expressions in your search string.

If a function panel exists for the function, LabWindows/CVI displays the panel. If two or more Function Panel windows exist for the function, LabWindows/CVI displays a list of the function panels.

The default shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

View»Find UI Object

Use the **Find UI Object** command to move directly from a Source window to the User Interface Editor. To use this command, place the cursor on the constant name or callback function name of the user interface panel, control, or menu object you want to view. Then select **View»Find UI Object**. LabWindows/CVI searches each `.uir` file that is currently open or in the project for user interface objects with a matching constant name or callback function name. If LabWindows/CVI finds an object, the User Interface Editor that contains the object comes to the foreground.

If the matching object is a panel, the panel title bar briefly flashes and the panel becomes active. If the object is a control, **Find UI Object** selects the control. If **Find UI Object** finds a menu object or more than one matching object, a dialog box that contains the list of matches

appears. In this dialog box you can view information about each of the objects or select one
to edit.

Build Menu for the Source and Interactive Execution Windows

This section explains how to use the commands in a Source or Interactive Execution window
Build menu. Use the commands in the **Build** menu to compile files and to build and link
projects.

Build»Configuration

This command works in the same way as the command in the **Build** menu in the Workspace
window. Refer to the section *Build Menu for the Workspace Window*, of Chapter 2,
Workspace Window, for more information.

Build»Compile File

You must compile your source code before executing it in a Source window or the Interactive
Execution window. The **Compile File** command adds the file to the project if necessary,
checks it for syntax errors, and compiles it. If the file has any build errors after
LabWindows/CVI completes compilation, the Build Errors window appears.

When you want to call a function that you define in a Source window from another Source
window, from the Interactive Execution window, or from a function panel, you must first
execute the **Compile File** command in the Source window where you define the function.
If you subsequently modify the function, you must recompile the Source window before
calling the function again.

You can specify compiler options in the Build Options dialog box.

Build»Create

The **Build»Create** command works in the same way as the command in the **Build** menu in
the Workspace window. Refer to the *Build Menu for the Workspace Window* section of
Chapter 2, *Workspace Window*, for more information.

Build»Mark File for Compilation

When LabWindows/CVI marks a source file for compilation, a C appears in the Status column
of the Project Tree in the Workspace window. LabWindows/CVI recompiles marked files the
next time you build the project. When you modify a source file, LabWindows/CVI
automatically marks the file for compilation. You can force LabWindows/CVI to compile a
source file on the next build by selecting **Mark File for Compilation**.

Build»Clear Interactive Declarations

Variables you declare in the Interactive Execution window remain in effect until you explicitly remove them. This feature lets you use these variables in succeeding executions of the Interactive Execution window and also enables different function panels to access the same variables.

When you delete the entire contents of the Interactive Execution window by selecting **Edit»Clear Window**, LabWindows/CVI removes the variables. If you want to remove the variables without deleting the contents of the Interactive Execution window, use the **Clear Interactive Declarations** command.

Build»Insert Include Statements

The **Insert Include Statements** command opens a dialog box you can use to select one or more header files to include at the top of the Source window.

Build»Add Missing Includes

If, when you last attempted to compile the source file, the compiler reported that function prototypes were missing, **Add Missing Includes** can find include (`.h`) files that contain some or all of the missing prototypes. This command inserts `#include` statements for these files into your source file at the current cursor position. LabWindows/CVI adds `#include` statements only for libraries or instrument drivers that appear in the **Instrument** or Library Tree, or **Library** menu.

Build»Generate Prototypes

After you compile a source file, you can use the **Generate Prototypes** command to generate a file that contains declarations for global and static functions and external declarations for global variables. The command generates the file into a new Source window. You can copy these declarations into your source and header files.

Build»Next/Previous Error/Item

If, when you compile a file or build your project, LabWindows/CVI displays multiple errors, you can use the **Next Error/Item** command to step to the next build error. LabWindows/CVI highlights source code as you step through the errors. You can use the **Previous Error/Item** command to step to the previous build error.

Build»Build Errors in Next File

If, when you build your project, LabWindows/CVI displays errors for multiple files, you can use the **Build Errors in Next File** command to step to your next file with build errors. LabWindows/CVI highlights source code as you step through the errors.

Run Menu for the Source and Interactive Execution Windows

This section explains how to use the commands in a Source or Interactive Execution window **Run** menu. Use the commands in the **Run** menu to run and debug programs. You can set breakpoints and watch expressions, step through code, view variable values, and more.

The **Debug**, **Continue**, **Step Over**, **Step Into**, **Finish Function**, **Terminate Execution**, **Break On**, **Breakpoints**, **Stack Trace**, **Up Call Stack**, **Down Call Stack**, **Execute**, **Command Line**, **Threads**, and **Loaded Modules** commands work in the same way as they do in the Workspace window. Refer to the *Run Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Introduction to Breakpoints and Watch Expressions

You can pause the execution of a program without aborting it altogether by marking breakpoints in code. You can use these breakpoints to interrupt program execution for debugging. Breakpoints can be either conditional or unconditional. Breakpoints apply to specific lines of code, but LabWindows/CVI maintains them separately from the source file. If you modify the source code outside of the LabWindows/CVI environment, you might invalidate breakpoint position information. If you set a breakpoint on a line that contains no executable code, a dialog box appears when you attempt to debug the project. This dialog box contains options to delete the breakpoint, disable the breakpoint, terminate debugging, or move the breakpoint to the next line that contains code. You can use the `Breakpoint` function to insert a breakpoint directly in the source file.

You also can use watch expressions for debugging. With watch expressions, you can specify that LabWindows/CVI suspend execution conditionally without regard to a specific line of code.

Note Breakpoints and watch expressions apply only to source code modules. You cannot set breakpoints in include files.

Note Some system functions on some systems might break execution when called with invalid arguments. For example, `CA_FreeMemory` might break execution when called with a pointer that was not allocated.

Breakpoint State

When a program reaches a breakpoint, LabWindows/CVI positions the keyboard cursor at the next program statement to execute and outlines the statement. You cannot edit the source code in the window while the breakpoint is in effect. However, you can look at other windows, change the state of breakpoints, and modify the value of variables in the Variables, Array

Display, and String Display windows. Also, if you are at a breakpoint in a Source window, you can execute code in the Interactive Execution window or in a function panel.

To resume program execution after a breakpoint, you have several options under the **Run** menu. You can restart the project at a breakpoint by selecting **Debug**. To halt the execution of a program at a breakpoint, select **Terminate Execution** while a LabWindows/CVI environment window is active.

Setting and Clearing Breakpoints

You can set and clear breakpoints in the following ways.

- If you select **View»Line Icons**, click the line icon area next to a line of code to set or clear a breakpoint on that line.

- Move the cursor to the line of code where you want to set or clear a breakpoint and select **Run»Toggle Breakpoint**.

- Select **Run»Breakpoints** to edit all breakpoints in the workspace and the Interactive Execution window. You also can use the **Breakpoints** command to set conditional breakpoints.

- Select **Run»Break On»First Statement** to break on the first executable statement in the project or the Interactive Execution window.

- You can set breakpoints directly in source code using the `Breakpoint` function.

- You can manually suspend execution while the program runs if the program checks for user input. For example, if the program makes calls to `RunUserInterface` or `scanf`, pressing <Ctrl-F12>, if you have default shortcut keys enabled, while a LabWindows/CVI environment window is active causes a breakpoint state.

Conditional Breakpoints

To set conditional breakpoints, select **Run»Breakpoints**. When you assign a conditional breakpoint to a line in a program, LabWindows/CVI evaluates an expression you supply, such as `x==100` or `y<0`, before executing the line. If the expression is true, program execution suspends.

For example, if you assigned these expressions to line 23 in a program, you would have to define `x` and `y` before line 23.

Watch Expressions

You can use watch expressions to suspend program execution conditionally. Watch expressions do not apply to specific lines of code. Instead, LabWindows/CVI evaluates them before each statement in source code.

Run»Run Interactive Statements

Select **Run Interactive Statements** in the Interactive Execution window to execute code in that window. You do not need to enter a complete program in the Interactive Execution window. For instance, you can execute variable declarations and assignment statements in C without declaring a `main` function.

You can use the Interactive Execution window to test portions of code before including them in the primary program. You also can use the Interactive Execution window to execute functions exported by a loaded instrument or by a file in the project if the project has been linked. The Interactive Execution window can access functions and data declared as global in a Source window, but a Source window cannot access the functions and data declared in the Interactive Execution window.

LabWindows/CVI does not disturb asynchronous I/O, RS-232 ports, opened files, and User Interface Library resources you use in the Interactive Execution window at the beginning or end of execution in the Interactive Execution window. LabWindows/CVI terminates, closes, or deletes these program elements only when you perform one of the following actions:

- Select **Build»Clear Interactive Declarations**.
- Select **Edit»Clear Window**.
- Link a project.
- Run a project.

Run-Time Error Reporting

LabWindows/CVI reports various run-time errors during the execution of a program. One example of a run-time error is a call to a LabWindows/CVI library function with an array or string that is too small to hold the output data.

When such errors occur, a dialog box appears identifying the type of error and the location in the file where the error occurred. LabWindows/CVI then displays the error in the Run-Time Errors window.

LabWindows/CVI suspends the program so you can inspect the values of variables in the Variables window. To terminate a program that suspended because of a run-time error, select **Run»Terminate Execution** or use the default shortcut key <Ctrl-F12> while a LabWindows/CVI environment window is active.

Run»Go to Cursor

When the program is in a breakpoint state, you can move the keyboard cursor to a line in the program and select **Run»Go to Cursor**. Program execution then continues until it reaches that line, where it enters another breakpoint state.

Run»Set Next Statement

Use the **Set Next Statement** command to change the next statement to execute while you are debugging. This command is useful if you want to skip over code you know is going to fail or if you executed code that failed and you want to go back and investigate that section.

To use this command, move the cursor to the statement you want to execute and select **Run»Set Next Statement**. A red outline appears surrounding the statement in the source code.

If you try moving from inside one function to another function or from one block or scope to another block or scope, LabWindows/CVI shows a warning dialog box. You can choose to continue or remain in the same position.

Run»Toggle Breakpoint

The **Toggle Breakpoint** command toggles the state of the breakpoint on the selected lines.

Run»View Variable Value

View Variable Value is a convenient way to view the contents of arrays, structures, and global variables that appear in source code. Highlight the variable that you want to see and select **View Variable Value**. Depending on the type of the variable, the Variables, Array Display, or String Display window appears with the selected variable highlighted.

Run»Add Watch Expression

Add Watch Expression is a convenient way to view the value of an expression that appears in source code. Highlight the expression that you want to see and select **Add Watch Expression**. The Watch window appears with the expression you selected.

Run»Evaluate Data Tooltip

The **Evaluate Data Tooltip** command displays the value of expressions, variables, and identifiers. If the value in the tooltip is bold, you can change the value by clicking the value. You also can hover the mouse over a value to view the tooltip.

Instrument Menu for the Source and Interactive Execution Windows

The **Instrument** menu for the Source and Interactive Execution windows works in the same way as the **Instrument** menu for the Workspace window. For more information about these commands, refer to the *Instrument Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Library Menu for the Source and Interactive Execution Windows

The **Library** menu for the Source and Interactive Execution windows works in the same way as the **Library** menu for the Workspace window. For more information about these commands, refer to the *Library Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Tools Menu for the Source and Interactive Execution Windows

This section explains how to use the commands in a Source and Interactive Execution window **Tools** menu.

The **Create ActiveX Controller**, **Create ActiveX Server**, **Edit ActiveX Server**, **Create IVI Instrument Driver**, **Create Instrument I/O Task**, **Create/Edit DAQmx Tasks**, **Source Code Control**, **Source Code Browser**, **User Defined Entries**, and **Customize** commands work in the same way as they do in the Workspace window. Refer to the *Tools Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Tools»Edit Instrument Attributes

Use the **Edit Instrument Attributes** command to add, delete, or edit attributes for an IVI instrument driver. You can select this command only if the file in the Source window has the same path and base filename as an instrument driver function panel (.fp) file and its associated .sub file. The command is useful only if you used the **Create IVI Instrument Driver** command to generate the instrument driver files.

The **Edit Instrument Attributes** command analyzes the instrument driver files to find all the attributes the driver uses. The command then opens a dialog box that displays the attributes and various information about them. In the dialog box, you can add or delete attributes, modify their properties, and enter help text for them. When you apply the changes, the command modifies the source, include, and function panel files for the instrument driver.

If you select this command when the text cursor is over the defined constant name or callback function name for one of the attributes, the dialog box appears with that attribute selected in the list box.

Refer to the *LabWindows/CVI Instrument Driver Developers Guide* for more information about the **Edit Instrument Attributes** command.

Tools»Edit Function Tree

Use **Edit Function Tree** command to display the Function Tree Editor window for the function panel (`.fp`) file associated with the file in the Source window. The function panel file must have the same path and base filename as the file in the Source window.

Tools»Edit Function Panel

Use the **Edit Function Panel** command to display the Function Panel Editor window for a function defined in an instrument driver source file. You can use this command only if the file in the Source window has the same path and base filename as an instrument driver function panel (`.fp`) file. The text cursor must be over the name of a function for which there is a function panel in the `.fp` file.

Window Menu for the Source and Interactive Execution Windows

The **Window** menu for the Source and Interactive Execution windows works in the same way as the **Window** menu for the User Interface Editor. For more information about these commands, refer to the *Window Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*.

Options Menu for the Source and Interactive Execution Windows

Use the commands in the **Options** menu to set up preferences for the LabWindows/CVI environment and execute various LabWindows/CVI utilities.

Options»Editor Preferences

Use the Editor Preferences dialog box to set up Source window editor preferences.

- **Undoable actions per file (next session)**—Set the number of actions per file that you can undo.

- **Purge undo actions when saving file**—Clear the accumulated list of editing actions each time you save a file.

- **Tab length**—Set the tab length. Activate the options to request LabWindows/CVI to convert tab characters into spaces when saving files and convert leading spaces to tab characters when loading files. These options are convenient if you use another editor or a printer that does not support tab characters.

- **Line terminator for this file** and **Line terminator for new files**—LabWindows/CVI can read source files with any of the commonly used line-termination sequences.

It remembers what line-termination sequence was found in each file and uses the same sequence when saving each file. If you want to change that sequence because you want to load the file into another editor, use the **Line Terminator** option as follows:

–   If you want to load a text file into DOS/Windows editors, select CR/LF termination.

–   If you want to load a text file into a Macintosh editor, select CR termination.

•   **Show function prototypes and struct/union fields while typing**—Select whether LabWindows/CVI shows the prototype of the function when you type the open parenthesis after a function name.

•   **Move cursor to end of pasted text**—Put the cursor at the end of the pasted text. Leave this option blank to put the cursor at the beginning of the pasted text.

Options»Toobar

Use this option to open the Customize Toolbars dialog box. For more information about customizing the toolbar, refer to the *Toolbars in LabWindows/CVI* section of Chapters 2, *Workspace Window*.

Options»Bracket Styles

Use the **Bracket Styles** command to set the location of curly brackets when the following commands generate them in programs.

•   **Edit»Insert Construct** in a Source window.

•   **Code»Generate** in the User Interface Editor.

You can specify two bracket styles: one for functions and another for statements, such as `if` and `switch` statements.

Options»Font

Use the **Font** command to select the font and font size for text in Source windows, Interactive Execution windows, and Variables windows. You can select from a list of monospace fonts.

Options»Colors

The **Colors** command in the Source and Interactive Execution windows works in the same way as the **Colors** command for the Workspace window. Refer to the *Options Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Options»Syntax Coloring

When you enable the **Syntax Coloring** command, LabWindows/CVI color codes the various types of tokens in your source and include files. LabWindows/CVI can color code the following types of tokens:

- C keywords
- Identifiers
- Comments
- Integers
- Real numbers
- Strings
- Preprocessor directives
- User-defined tokens

To set the color for a token type, select **Options»Colors**.

To create the list of user-defined tokens, select **Options»User Defined Tokens for Coloring**.

Options»User Defined Tokens for Coloring

The **User Defined Tokens for Coloring** command opens a dialog box in which you can define tokens for display in a unique color when you enable the **Syntax Coloring** command. Use the **Colors** command to set the color. Each token must be in the form of a valid C identifier. For each token, you can choose whether to save it in the workspace file or from one LabWindows/CVI session to another.

Options»Translate LW DOS Program

Use the **Translate LW DOS Program** command to convert a source file written in LabWindows for DOS so that it can run in LabWindows/CVI.

Options»Generate DLL Import Source

This command generates source code that you can use to create a DLL import library. In general, it is not necessary to use this command. For most cases, you can generate a DLL import library directly using the **Generate DLL Import Library** command. Use this command only when you must perform special processing in the DLL import library. LabWindows/CVI never requires such special processing.

LabWindows/CVI enables the **Generate DLL Import Source** command only when you have an include file in the Source window. The include file must contain declarations of all the DLL functions you want to access. When you execute the command, a dialog box appears in which you enter the pathname of the DLL.

The **Generate DLL Import Source** command generates the import library source into a new Source window. You can modify the code, including making calls to functions in other source files. Create a new project that contains the source file and any other files it references. Select **Build»Target Type»Static Library** in the Workspace window. Execute the **Create Static Library** command.

Note   You cannot export variables from a DLL using the import library source code this command generates. When you want to export a variable, create functions to get and set its value or create a function to return a pointer to the variable.

Note   When you edit the source code this command generates, you cannot use the import qualifier in the function declarations in the DLL include file.

Note   The import source code does not operate in the same way as a normal DLL import library. When you link a normal DLL import library into an executable, the operating system attempts to load the DLL as soon as the program starts. The import source code operates in such a way that the DLL does not load until the user makes the first function call into it.

Options»Generate DLL Import Library

This command generates a DLL import library. LabWindows/CVI enables **Generate DLL Import Library** only when you have an include file in the Source window. The include file must contain declarations of all the functions and global variables you want to access from the DLL. When you execute the command, a dialog box appears giving you the option to generate an import library for each of the compatible external compilers rather than just for the current compatible compiler. Enter the pathname of the DLL in the file dialog box that appears.

The **Generate DLL Import Library** command generates a .lib file with the same base filename as the include file. If you choose to create an import library for both compilers, LabWindows/CVI creates the files in subdirectories named msvc and borland. LabWindows/CVI creates a copy of the library for the current compatible compiler in the directory of the DLL.

Options»Generate Visual Basic Include

This command generates a Microsoft Visual Basic include file from the .h file of an instrument driver. Use this command if you are porting an instrument driver to a DLL for use in Microsoft Visual Basic.

Options»Generate Function Tree

Use this command to generate a function tree from a header file. This command opens a dialog box, which contains the following options:

- **Name**—The name of the instrument to generate.

- **Prefix**—The common prefix for functions in the instrument. For example if you have the following functions in the header file

```
int DllExport __CFUNC MyInstrumentInit (void);
int DllExport __CFUNC_C MyInstrumentFormat (char *format, ...);
int DllExport __CFUNC MyInstrumentClose (void);
```

  you should use `MyInstrument` as the prefix.

- **Default Qualifier**—The default qualifier for functions in the instrument. LabWindows/CVI will include in the generated function tree only the functions with the default qualifier you specify or the specified default qualifier appended with `_C`. For example, for the preceding functions, you should use `DllExport __CFUNC` as the default qualifier.

- **FP File**—The absolute path of the generated function panel file.

- **Header File**—The absolute path of the generated header file. This header file is identical to the one used for generating the function tree except that all the special tags are removed. This field is optional.

LabWindows/CVI creates a function panel for each function. You must follow specific rules and use special tags in the header file.

When you select this option, LabWindows/CVI performs the following actions for each function panel it creates.

- Removes the function prefix

- Capitalizes the first letter

- Inserts a space before each first capital letter or digit in a group of capitals or digits

  For example, `Set6100SensorRpms` becomes `Set 6100 Sensor Rpms`

- Replaces substrings, specified with `XCHG` and `PXCH`, in function names and parameter names

- Places the function in the function tree under the current class and with the current parameters

Rules for Header Files

Your header files must conform to the following rules:

- Declare all non-LabWindows/CVI types in the header file with `typedef`. `#define` for a type does not work.

- For enums, keep the following rules in mind:

    – Declare one enum per line.

    – Use a comment (`//`) after the enum to be used as a value label in the function panel.

    – Do not assign values. LabWindows/CVI ignores assigned values.

    – Do not use additional comments in enums; the `//` comment specifies an optional label for each enum value.

    – Avoid semicolons inside enumeration lists, including comments.

- Use `const char string[]` for input strings.

- Make sure that function declarations contain the specified function qualifier. For example, if you specified `CVIFUNC` as function qualifier, every function must start with `CVIFUNC` or `CVIFUNC_C`.

- Avoid nested comments or combination of comment styles (`//`, `/**/`, or `/* // */`).

- Do not use nested structure definitions. Multiple bracket levels are not recognized.

- Do not use comments inside function declarations.

Tags in Header Files

Use the following tags in the header file.

Tag	Description
`/// ; Comment here`	Ignores the whole line as a comment.
`/// -> Class Name`	Creates a new class node of the specified name.
`/// <- Class Name`	Goes one level up/back in the function tree. The class name is optional and is used for better orientation in the header file.

`/// OUT p1,...`	Specifies which parameter of a function is output. This tag applies only to the function immediately following the tag.
	
	Example use
	`/// OUT 2,3,4`
	`void SplitPath (char Pathname[], char Drive_Name[], char Directory_Name[], char File_Name[]);`
	
	Note If you do not use an `OUT` tag, any parameter with a `*` or `[]` type for functions starting with `Get` is considered an output parameter. If you use an `OUT` tag, only those parameters you specify in the `OUT` tag are output parameters.

Tag	**Description**
`/// PTYP p1/type,...`	Changes the parameter type for the specified parameters. This tag applies only to the function immediately following the tag. Example use `/// PTYP 2/Any Type,4/Any Array,5/Numeric Array`
`/// DFLT p1/val,...`	Specifies the default value for a parameter. This tag applies only to the function immediately following the tag.
`/// CUST p1/dll/fn,...`	Specifies the customization DLL and handler for a parameter. This tag applies only to the function immediately following the tag.
`/// EHDV p1/value,...`	Specifies which enum has an additional default value. This tag applies only to the function immediately following the tag. A default value is added to the beginning of the enum. You can use this tag with RNG and SLD tags.
`/// ENUM name`	Starts a list of #define values that represents an enum of the specified name. You can use this tag with RNG, BIN, and SLD tags. The list of values must end with an empty line.

`/// BIN p1/enum,...`	Specifies which parameter of a function is a binary control. This tag applies only to the function immediately following the tag. *enum* is optional; it specifies the enum name. *enum* is the name of a regular enum or an enum created using the ENUM tag. The parameter must be of an enum type if you omit the slash. The enum must contain exactly two values.
	
	Example use
	`/// BIN 2`
	`void OpenFile(char Pathname[], enumTxtBin mode);`
	
	Example use
	`/// BIN 2/enumTxtBin`
	`void OpenFile(char Pathname[], unsigned mode);`

Tag	Description

`/// SLD p1/enum,...`	Specifies which parameter of a function is a slide control. This tag applies only to the function immediately following the tag. *enum* is optional; it specifies the enum name. *enum* is the name of a regular enum or an enum created using the ENUM tag. The parameter must be of an enum type if you omit the slash. The enum must contain at least two values.
	
	Example use
	`/// SLD 2`
	`void OpenFile(char Pathname[], enumTxtBin mode);`
	
	Example use
	`/// SLD 2/enumTxtBin`
	`void OpenFile(char Pathname[], unsigned mode);`

`/// RNG p1/enum,...`	Specifies which parameter of a function is a ring control. This tag applies only to the function immediately following the tag. *enum* is optional; it specifies the enum name. *enum* is name of a regular enum or an enum created using the ENUM tag. The parameter must be of an enum type if you omit the slash. The enum must contain at least one value.
	
	Example use
	`/// ; You can omit the RNG tag in this case.`
	`/// ; The ring is created automatically if an`
	`/// ; enum is found in the place of the parameter`
	`/// ; type.`
	`/// RNG 2`
	`void OpenFile(char Pathname[], enumTxtBin mode);`
	
	Example use
	`/// RNG 2/enumTxtBin`
	`void OpenFile(char Pathname[], unsigned mode);`

`/// ERNG p1,...`	Specifies which parameter of a function is an empty ring control. This tag is mostly used for Set/GetAttribute functions. This tag applies only to the function immediately following the tag. The parameter type cannot be of an enum type.

Tag	Description
`/// XCHG s1/s2,...`	Replaces string `s1` with `s2` in function panel titles. `/// XCHG` stops any replacements. For example, if you have a function with a name `PrefixVXIPnPCfg`, the generated title for this function is `V XIPn PCfg`, because of the word splitting mechanism. You can correct problems like this by using the XCHG tag. For example, place the following XCHG tag in front of the function in your header file: `/// XCHG V XIPn PCfg/VXI PnP Configuration`. Your function panel title will then be `VXI PnP Configuration`. This tag applies only to the function immediately following the tag.
`/// PXCH s1/s2,...`	Replaces string `s1` with `s2` in function panel controls. `/// PXCH` stops any replacements.
`/// BINB type/OffVal/OnVal/ defVal`	Specifies what data type to associate with a binary switch control. This tag also specifies labels for both values and index of the default value. This tag is valid until it is overridden by another or no data type. You cannot use this tag for an enum type. Example use `/// BINB boolean/Off/On/1` causes every Boolean parameter to generate a binary switch with labels `Off` and `On` with a default value of `On`. ✎ **Note** NI recommends that you use the BINB tag only for Boolean parameters.

`/// VARG type/name,...`	Adds specified parameters at the end of the parameter list that ends with ... (variable argument list). This tag applies only to the function immediately following the tag.
	
	Example use
	`/// VARG Any Type/Value,int/Number of Elements`
	
	You optionally can specify the name of the help file whose contents are copied into the control help for name. Use the following tag:
	
	`/// VARG type/name/helpfile,...`

Tag	Description
`/// RETV name`	Specifies the name of the return value. If you do not use this tag or use the tag without a name, the default name `Return value` is used.
`/// FP title`	Specifies the title of the instrument. If you do not use this tag, the function panel filename is used for the function panel title.

`/// ADDT typename`	Adds a type to the function panel.
	
	Adds: `typename`
	`typename []`
	`typename *`
	`typename **`
	`typename *[]`

`/// EX`	Excludes the following lines from function panel generation. This tag ignores all other tags, functions, enums, and so on until it finds the UNEX tag. **Note** The code excluded from the generation still has to follow the syntax. If you want to exclude code that is not syntactically correct, use comments instead. Also, EX and UNEX tags support nesting.
`/// UNEX`	Stops excluding code from function panel generation. This tag supports nesting.
`/// HFP helpfile`	Copies the contents of `helpfile` into the instrument help of the function tree.
`/// HCL helpfile`	Sets the class help for the class node following this tag to the contents of `helpfile`. This tag is overwritten by the first class node following this tag.
`/// HFUN helpfile`	Sets the function help for the function following this tag to the contents of `helpfile`. This tag applies only to the function immediately following the tag.
`/// HPAR p1/` `helpfile1,...`	Sets the help for parameter **p1** for the function following this tag to the contents of `helpfile`. This tag applies only to the function immediately following the tag.
`/// HRET helpfile`	Sets the help for the return value for the function following this tag to the contents of `helpfile`. This tag applies only to the function immediately following the tag.

Options»Create Object File

You can use the **Create Object File** command to compile the contents of a Source window into an object file. Compiled files consume less memory and run faster than source files. Object files are especially useful for instrument driver programs because they load faster. Compiled files cannot be debugged, however, and they do not have run-time error checking.

Note   If the source file is in the project and you do not want to debug it, use **Enable 'O' Option** in the Workspace window rather than the **Create Object File** command. To select **Enable 'O' Option**, right-click the source file in the Project Tree and select **Enable 'O' Option**.

The **Create Object File** command gives you the option of creating an object file for both of the compatible external compilers rather than just for the current compatible compiler. If you choose to create an object file for both compilers, LabWindows/CVI creates the files in subdirectories named `msvc` and `borland`. LabWindows/CVI creates a copy of the object file for the current compatible compiler in the directory of the source file.

You can compile your file using a third-party compiler that LabWindows/CVI supports. These compiled files are smaller and execute faster than object files LabWindows/CVI creates. You can use the **Create Object File** command if you do not have access to another compiler.

Options»Preprocess Source File

Use the **Preprocess Source File** command to open a new source window that contains preprocessed output. LabWindows/CVI replaces simple macros, expands function macros, includes header files, and resolves conditional compilation.

Help Menu for the Source and Interactive Execution Windows

The **Help** menu for the Source and Interactive Execution windows works in the same way as the **Help** menu for the Workspace window. For more information about these commands, refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Help»Keyboard Help

This command shows a list of default shortcut keys you can use in the Source window.

5

Using Instrument Drivers

This chapter presents a general overview of instrument drivers. Refer to the *LabWindows/CVI Instrument Driver Developers Guide* for more information about creating instrument drivers.

An instrument driver is a set of high-level functions with graphical function panels that make programming easier. The driver encapsulates many low-level operations, such as data formatting and communication with GPIB, RS-232, and VXI instruments, into intuitive, high-level functions. An instrument driver usually controls a physical instrument but also can serve as a software utility.

Instrument driver programs have an associated include file that declares the high-level functions you can call, the global variables you can access, and the defined constants you can use.

Instrument Driver Files

A LabWindows/CVI instrument driver typically consists of the following three or four files. Each file has the same base filename, which is typically an abbreviation of the actual instrument name. The instrument driver files must reside in the same directory on your disk, or they must be in the appropriate VXI*plug&play* directories or in the appropriate IVI directories.

- The function panels are in a `.fp` file.

- For instrument drivers that use an attribute model, such as IVI drivers, there can be an additional `.sub` file that contains attribute information displayed in the function panels.

- The function, variable, and defined constant declarations are in a `.h` include file.

- The instrument driver program can be in one of several different types of files.

  - A `.c` source file.

  - A `.obj` object file that contains one or more compiled C modules or a `.lib` library file. You must compile these files in LabWindows/CVI or in a compatible external compiler.

For example, the instrument module files for a Fluke 8840A multimeter are `fl8840a.fp`, `fl8840a.c`, and `fl8840a.h`.

You can load an instrument driver into the LabWindows/CVI interactive program whether the instrument program is in the form of a `.c`, `.obj`, or `.lib` file. The presence of the `.h` file is

essential because you must include it in your program to reference functions, global variables, and constants in the instrument driver.

VXI*plug&play* Instrument Driver Files

When you install a VXI*plug&play* instrument driver, the installation program does not place the `.h` file in the same directory as the `.fp` file. The installation program places the `.h` file in the `VXIPNP\include` subdirectory. LabWindows/CVI can then find the include files in this subdirectory.

When you install a VXI*plug&play* instrument driver, the installation program places the `.c` file in the same directory as the `.fp` file. The installation program also installs a dynamic link library (`.dll`) and two import libraries (`.lib`), one compatible with Visual C/C++ and the other compatible with Borland C/C++. These files are not in the same directory as the `.fp` file. The import libraries are in the `VXIPNP\lib\msc` and `VXIPNP\lib\bc` subdirectories. The DLL is in the `VXIPNP\bin` subdirectory. The installation program adds the `VXIPNP\bin` directory to the `PATH` environment variable so that the DLL can be found using the standard Windows DLL search algorithm.

If the `.fp` file is under the VXI*plug&play* framework directory, LabWindows/CVI can find the appropriate import library. If LabWindows/CVI finds the import library, it gives it precedence over the `.c` file as the program file for the instrument driver.

IVI Instrument Driver Files

When you install an IVI instrument driver, the installation program does not place the `.h` file in the same directory as the `.fp` file. The installation program places the `.h` file in the `IVI\include` subdirectory. LabWindows/CVI then can find the include files in this subdirectory.

When you install an IVI instrument driver, the installation program installs a dynamic link library (`.dll`) and two import libraries (`.lib`), one compatible with Visual C/C++ and the other with Borland C/C++. These files are not in the same directory as the `.fp` file. The import libraries are in the `IVI\Lib\msc` and `IVI\Lib\bc` subdirectories. The DLL is in the `IVI\bin` subdirectory. The installation program adds the `IVI\bin` directory to the `PATH` environment variable so that the DLL can be found using the standard Windows DLL search algorithm.

If the `.fp` file is under the IVI framework directory, LabWindows/CVI can find the appropriate import library. If LabWindows/CVI finds the import library, it gives it precedence over the `.c` file as the program file for the instrument driver.

Loading/Unloading Instrument Drivers

You can load and unload instrument drivers manually using the Instruments folder in the Library Tree or using the **Instrument** menu. Instrument drivers loaded through the Instruments Folder or **Instrument** menu do not have to be listed in the project, and you can load or unload them at any time except during program execution. You also can load instrument drivers by dropping `.fp` files onto the Library Tree directly.

To incorporate instrument drivers into the project, select **File»Add to Project** in a Function Panel window or the Function Tree Editor or select **Edit»Add Files to Project** in the Workspace window. If you drop the `.fp` file onto a project in the Project Tree, LabWindows/CVI adds the file to the project and loads the instrument driver if the project is the active project. If you drop the `.fp` file onto a non-active project, LabWindows/CVI adds the `.fp` to that project, but does not load the `.fp`. If you drop the `.fp` file onto the Library Tree, LabWindows/CVI loads the instrument driver but does not add it to any project. The `.fp` file represents the instrument driver in the project. If the `.fp` file is in the project when you open the project, LabWindows/CVI automatically loads the instrument driver and removes it when you unload or deactivate the project.

Precedence Rules for Loading the Instrument Driver Program File

When you load a `.fp` file, LabWindows/CVI loads the instrument driver program file. In some cases, you might have an instrument driver program file in more than one format. For instance, you might have `fl8840a.obj` and `fl8840a.c` in the same directory, which can occur when you obtain the source code for the instrument driver and then compile it into an object file. LabWindows/CVI chooses which file to load according to the following rules:

- If an instrument driver program file is in the project, LabWindows/CVI loads it. There can be at most one unexcluded program file with the same base name as the `.fp` file in the project list. Thus, x`.obj` and x`.c` cannot be in the project list at the same time unless you exclude one or both of them.

- If both of the following conditions apply with VXI*plug&play* instrument drivers, the `.lib` file is associated with the `.fp` file.
  - The `.fp` file is under the VXI*plug&play* framework directory.
  - A `.lib` file is in the appropriate VXI*plug&play* framework subdirectory. The following table shows the corresponding subdirectories.

Compatible Compiler	Corresponding Subdirectory that Contains the .lib File
Visual C/C++	`lib\msc`
Borland C/C++ /C++ Builder	`lib\bc`

–   If both of the following conditions apply with IVI instrument drivers, the `.lib` file is associated with the `.fp` file:

–   The `.fp` file is under the IVI framework directory.

–   A `.lib` file is in the appropriate IVI framework subdirectory. The following table shows the corresponding subdirectories.

Compatible Compiler	Corresponding Subdirectory that Contains the .lib File
Visual C/C++	`lib\msc`
Borland C/C++/C++ Builder	`lib\bc`

•   If an instrument driver program file is on disk in the same directory as the `.fp` file, LabWindows/CVI loads it with the following precedence:

1.   `.lib`
2.   `.obj`
3.   `.c`

Loading an Instrument without an Instrument Program

You can load a `.fp` file as an instrument, even if no program file exists for it. In this case, LabWindows/CVI does not associate a program with the `.fp` file. Nevertheless, the `.fp` file appears in the Instruments folder and the **Instrument** menu.

Loading a `.fp` file without a program is useful if you want to use `.fp` files to document functions in your project. When you do not provide a program file for the `.fp` file, you cannot execute the function panels, but you can insert code into Source windows from them.

If you try to execute an instrument driver function panel without an associated program, LabWindows/CVI reports a run-time error. You can attach a `.c` file to a `.fp` file after you load the `.fp` file or run the program file search algorithm in the Edit Instrument dialog box by selecting **Attach and Edit Source**.

Modules That Contain Non-Instrument Functions

Although the LabWindows/CVI instrument driver mechanism is primarily for program modules that control instruments, you can use it for any module that contains a set of high-level functions.

Suppose, for instance, you write a set of specialized analysis functions. If you develop function panels and a `.h` file for the module, you can load the module from the **Instrument** menu or Instruments folder in the Library Tree and call the functions from the function panels.

Modifying an Instrument Driver

You might want to modify an instrument driver that you received from National Instruments or elsewhere. If you want to modify the instrument driver program file, you must have the `.c` file for the instrument driver.

Before modifying an instrument driver, familiarize yourself with the *LabWindows/CVI Instrument Driver Developers Guide*.

You can modify the following four parts of an instrument driver:

- You can modify the function tree by selecting the `.fp` file using the **File»Open»Function Tree (*.fp)** command, selecting **Instrument»Edit**, right-clicking the Instruments folder and selecting **Edit Instrument**, or selecting **Tools»Edit Function Tree** from a Source window that contains the instrument driver source or include file.

- You can modify the function panels by selecting **Options»Edit Function Panel** from a Function Panel window, selecting **Edit»Edit Function Panel Window** from a Function Tree Editor window, or selecting **Tools»Edit Function Panel** from a Source window that contains the instrument driver source or include file when the text cursor is over the name of the function in the driver.

- You can modify the instrument driver program file by selecting **Instrument»Edit** in a Function Panel window, selecting **Tools»Go to Definition** from a Function Panel Editor window, or selecting **Go to Definition** from the context menu in the Function Tree Editor.

- You can modify the instrument driver include file by selecting the `.h` file using the **File»Open»Include (*.h)** command, by selecting **Tools»Go to Declaration** from a Function Panel Editor window, or by selecting **Go to Declaration** from the context menu of the Function Tree Editor.

Building IVI Instrument Drivers

For information about creating an IVI instrument driver, refer to the *LabWindows/CVI Instrument Driver Developers Guide*.

Fundamentals Overview

This section covers instrument driver fundamentals you must consider if you add functions to a driver, or if you develop a driver without using the Instrument Driver Development Wizard.

Although the wizard is the recommended method for developing a driver for an instrument that you control through a GPIB, VXI, or serial interface, there may be situations in which you develop a driver for other purposes. For example, you might develop a driver for common utility functions or analysis algorithms or a driver for controlling custom hardware that does not fit the IVI instrument driver model for GPIB or VXI devices.

Defining the Instrument Functions

An instrument driver exports a set of functions that programmers can use to control an instrument. To make the set of functions easy to use, you must organize them into a logical structure. Group related functions into categories or classes. For complex instruments, group related classes into higher-level classes. For very complex instruments that incorporate multiple personalities, you might consider creating multiple instrument drivers.

Structuring Functions in an Instrument Driver

If you use the wizard with a predefined instrument template, the wizard creates a function hierarchy for you. Otherwise, you must define and structure the driver functions on your own.

The three implementations of a single instrument driver hierarchy in this section illustrate options for structuring functions. In this example, the driver includes seven functions with which to program the instrument.

The first implementation gives the user a simple linear list of all available functions.

```
instrumentA
    function1
    function2
    function3
    function4
    function5
    function6
    function7
```

The second implementation breaks the functions into two function classes.

```
instrumentA
    function_class1
        function1.1
        function1.2
        function1.3
        function1.4
    function_class2
        function2.5
        function2.6
        function2.7
```

The third implementation treats the two function classes as two distinct instruments.

```
instrumentA
    function1
    function2
```

```
    function3
    function4
instrumentB
    function5
    function6
    function7
```

To successfully structure the functions for your instrument, you must determine who will use the instrument driver and how they will use the instrument. Define functions that stand alone to perform a useful action. For example, it might at first seem logical to use the functions `SetDMMRange` and `SetDMMFunction` for setting the range and function of a multimeter. However, a more useful function might be `ConfigureMeasurement`, for setting up multiple parameters.

Defining the Hierarchy of Functions

It is important to design the function hierarchy for the instrument driver carefully. When you do, the user can identify the functions required by the desired action without the burden of choosing from a long list of unrelated functions.

The concept of function classes is only apparent to the user from within the LabWindows/CVI development environment. The application program calls all functions within an instrument driver the same way, regardless of which function class they are in.

Defining the Function Parameters

To design the code for an instrument driver function, you must first establish its parameters.

Function parameters provide input information to the function and output variables where the function can store its results. Output parameters often contain values that the function reads from the instrument and formats for the user.

Data Types

You must specify a data type for each parameter in each instrument driver function. All data types the instrument driver uses must be intrinsic C data types or data types that you define in the `.h` file and list in the `.fp` file. Specify the data type of a parameter when you create its corresponding control on a function panel. This data type also must be consistent with the function prototypes in the instrument driver header file.

LabWindows/CVI uses the data type information to implement the variable declaration and run-time checking capabilities when users operate function panels. When you declare a variable from a function panel, LabWindows/CVI presents options based on the data type you specify for the function panel control. When you run a function from a function panel,

LabWindows/CVI verifies that the data type of the value you enter in the control matches the prototype of the function.

Data types are divided into three classes: predefined data types, user-defined data types, and VISA data types.

Predefined Data Types

Predefined data types are available by default in the LabWindows/CVI environment. The predefined data types consist of intrinsic C data types and meta data types that LabWindows/CVI defines.

Intrinsic C Data Types

The intrinsic C data types that LabWindows/CVI defines are as follows:

```
int
long
short
char
unsigned int
unsigned long
unsigned short
unsigned char
int []
long []
short []
char []
unsigned int []
unsigned long []
unsigned short []
unsigned char []
double
float
double []
float []
char *
char *[]
void *
```

When you create a control to represent an array of data, make the data type an intrinsic C data type that ends with the square brackets, []. Do *not* select a data type that ends with an asterisk (*). The brackets tell LabWindows/CVI that the control represents an array of data, not a pointer. LabWindows/CVI can then perform the appropriate variable declaration and run-time checking when the user operates the function panel.

When you define a function panel control with an intrinsic C data type, variables the user declares in the control through the **Declare Variable** command appear with that data type in the dialog box. You must define the parameter with the same data type when you prototype the function in the instrument driver include file.

Meta Data Types

The meta data types are useful for parameters through which users can pass arguments of more than one data type. The meta data types are `Numeric Array`, `Any Array`, `Any Type`, and `Var Args`. Each of these data types defines a set of multiple allowable data types. When the user executes the **Declare Variable** command on a control defined with a meta data type, the user can select from a list of the allowable data types.

Numeric Array

`Numeric Array` specifies a parameter that can be any of the intrinsic C numeric array data types. You must define the parameter as `void *` in the function prototype. An example of a `Numeric Array` data type is in the `PlotX` function of the User Interface Library. The `PlotX` function plots the values of any intrinsic C numeric array data type to a graph control on a user interface panel. On the function panel, the **X Array** control is of type `Numeric Array`. **X Array** is defined as `void *` in the following function prototype:

```
int PlotX (int panel, int control, void *xArray, int numPoints, int
    xDType, int plotStyle, int pointStyle, int lineStyle, int
    pointFreq, int color);
```

Any Array

`Any Array` specifies a parameter that can be any of the intrinsic C or user-defined array data types. You must define the parameter as `void *` in the function prototype. An example of an `Any Array` data type is in the `memcpy` function of the ANSI C Library. This function copies a specified number of bytes to a target buffer of any type from a source buffer. In the function panel, the first parameter is **Target Buffer**, which is of type `Any Array`. **Target Buffer** is defined as `void *` in the following function prototype:

```
void *memcpy (void *, const void *, size_t);
```

Any Type

`Any Type` specifies a parameter that can be any of the intrinsic C or user-defined data types. If the parameter is an output parameter, you must define it as `void *` in the function prototype. If the parameter is an input parameter, you must define it as `. . .` in the function prototype, and it must be the last parameter in the function. The **Value** output parameter of the `GetCtrlAttribute` function in the User Interface Library is an example of the `Any Type` data type. The function obtains the value of a particular attribute for a particular user interface control. Different attributes have different data types. Users pass the attribute value

parameter by reference, and it is therefore a pointer. Consequently, the attribute value parameter is of type `Any Type` and is defined as `void *` in the following function prototype:

```
int GetCtrlAttribute(int panel, int control, int attribute, void
    *value);
```

The **Value** parameter of the `SetCtrlAttribute` function also applies to attributes of different data types, but it is an input rather than an output parameter. Users pass this parameter by value rather than by reference, and thus it can have different sizes. For example, it might be an `int` or a `double`. Consequently, the attribute value parameter is of type `Any Type` and is defined as `...` in the following function prototype:

```
int SetCtrlAttribute (int panel, int control, int attribute, ...);
```

Var Args

`Var Args` specifies a variable number of parameters that can be any of the intrinsic C or user-defined data types. You must define the parameters as `...` in the function prototype. The `printf` and `scanf` functions in the ANSI C Library have examples of the `Var Args` data type. Following the **format string** parameter in each function, you can specify one or more parameters of different data types to match the type specifiers in the format string. In `printf`, the parameters are passed by value. In `scanf`, they are passed by reference and thus are really pointers. For both functions, one `Var Arg` function panel control is used, and `...` appears in the following function prototypes:

```
int printf (const char *, ...);
int scanf (const char *, ...);
```

User-Defined Data Types

You also can define data types and use them in function panels. You must declare user-defined data types in the function panel file of an instrument driver, and you must define the data type in the include file for the driver. Declare user-defined data types with the **Data Types** option in the Function Panel Editor.

For example, you can define a `waveform_var` data type for an instrument driver to represent waveform data. This `waveform_var` data type could be a structure that contains an array of doubles to represent the individual points in the waveform, a `float` for the time of the first point, and a `float` for the time between points.

Creating a User-Defined Data Type

Complete the following steps to create a user-defined data type for use in a function panel.

1. Define the data type with a `typedef` statement in the instrument driver header file.

    Using the previous example of the `waveform_var` data type, include the following code in the include file for the instrument driver.

```
typedef struct {double waveform_arr [500]; float t_zero; float
    t_delta; } waveform_var;
```

2. Add the data type to the instrument driver function panel file using the **Options»Data
   Types** command in the Function Panel Editor.

   Make the `waveform_var` data type available in the function panel file. Select
   Options»Data Types in the Function Panel Editor and enter `waveform_var` in **Type** of
   the Edit Data Type List dialog box. Click **Add**.

Now you can select the `waveform_var` data type when you create function panel controls for
this instrument driver. Also, users can interactively declare a variable of `waveform_var` data
type from any function panel control that you define as `waveform_var`.

User-Defined Array Data Types

Use care when you declare user-defined data types that are arrays. If you want to define a
user-defined array data type, square brackets `[]` must appear at the end of the type name in
the Edit Data Type List dialog box. The brackets enable the interactive variable declaration
and other capabilities of LabWindows/CVI function panels. For example, to declare an array
of `waveform_var` type from the preceding example, add `waveform_var []` in **Type** of the
Edit Data Type List dialog box and include the `typedef` declaration for `waveform_var` in
the driver include file. This example is correct because it includes brackets.

The following examples show incorrect ways to define user-defined array data types.

Assume the following data type definitions are in an instrument driver header file.

```
typedef waveform_var * waveform_arr1;
typedef waveform_var waveform_arr2[100];
```

The following data type declarations in the Edit Data Type List dialog box are incorrect.

```
waveform_var *
```
(This example is incorrect because it lacks brackets.)
```
waveform_arr1
```
(This example is incorrect because it lacks brackets.)
```
waveform_arr2
```
(This example is incorrect because it lacks brackets.)

VISA Data Types

The VISA Library defines a special set of data types. The IVI Library also uses some of these
data types. The data types strictly define the type and size of the parameters and therefore
promote the portability of the functions to new operating systems and programming
languages.

A subset of the VISA data types has been defined for use in the development of
LabWindows/CVI instrument drivers and are accessible as user-defined data types. The
following table shows these special data types for instrument drivers.

VISA Type Name	Definition
ViInt16	Signed 16-bit integer
ViInt32	Signed 32-bit integer
ViUInt16	Unsigned 16-bit integer
ViUInt32	Unsigned 32-bit integer
ViReal64	64-bit floating-point number
ViInt16[]	An array of ViInt16 values
ViInt32[]	An array of ViInt32 values
ViReal64[]	An array of ViReal64 values
ViChar[]	A string buffer
ViConstString	A read-only string
ViRsrc	An instrument driver resource descriptor (string)
ViSession	An instrument driver session handle
ViStatus	An instrument driver return status type
ViBoolean	Boolean value
ViBoolean[]	An array of ViBoolean values

To use these special user-defined data types in an instrument driver, complete the following steps:

1. Add the VISA data types to the function panel file by using the **Options»Data Type** command in the Function Panel Editor. Then click the **Add VISA Types** button in the Edit Data Type List dialog box.

2. Include the vpptype.h file in the instrument driver header file.

Input and Output Parameters

Because most instrument drivers are designed to control a physical instrument, the input and output function parameters often correspond to one or more of the controls on the face of the instrument.

Define input and output parameters as follows:

1. Review the purpose of the function to determine the inputs and outputs.

2. Choose the data type of each parameter. The data type should be one that the application program can use easily.

   • If a parameter is an array data type, select a data type with square brackets `[]` at the end of the data type name.

   • For output parameters, select the data type of the value that users pass by reference, not the pointer to that data type. When users operate function panels interactively, LabWindows/CVI knows to pass a variable by reference because the control is defined as an output.

     For example, if a function by the name of `examp_func` has an **examp_out** integer output parameter, prototype the function in the instrument driver include file as

     `examp_func (int *examp_out);`

     When you create a function panel for this function, create an output control for **examp_out** and specify its data type as `int`, not as `int *`. When a user declares variables interactively from the function panel, LabWindows/CVI creates an `int` variable and automatically puts an ampersand (&) in front of the variable name to pass it by reference.

3. Assign a meaningful name to each parameter.

Return Values

Instrument driver functions also can have a return value. Instrument drivers supplied by National Instruments use function return values to implement an error handling mechanism. All instrument driver functions have a return value of type `ViStatus` (32-bit unsigned integer) that returns error and status information about the function call.

Required Instrument Driver Functions

All instrument drivers must contain functions that perform the following operations:

• *Prefix*_init

• *Prefix*_close

• *Prefix*_error_message

The VXI*plug&play* standard requires the following additional functions for instrument drivers that control GPIB, VXI, or serial instruments:

• *Prefix*_reset

• *Prefix*_self_test

• *Prefix*_revision_query

• *Prefix*_error_query

IVI instrument drivers must have the following additional functions:

- *Prefix*_InitWithOptions
- *Prefix*_GetErrorInfo
- *Prefix*_ClearErrorInfo
- *Prefix*_LockSession
- *Prefix*_UnlockSession
- *Prefix*_ReadInstrData
- *Prefix*_WriteInstrData

IVI instrument drivers from National Instruments also must export functions by the name of *Prefix*_IviInit and *Prefix*_IviClose, but the driver must *not* have function panels for these functions. The *Prefix*_init function calls the *Prefix*_InitWithOptions function, which in turn calls the *Prefix*_IviInit function. The *Prefix*_close function calls the *Prefix*_IviClose function.

Building the Function Tree

Use the Function Tree Editor to create the function tree for your instrument driver. The function tree shows the hierarchy of instrument driver functions. Users can access functions from the function tree.

The function tree also contains help that the user can access from the dialog boxes. Add help as you create the tree.

Building the Function Panels

Users operate the function panels to execute instrument driver functions and to generate code for an application program. Each primary function requires a function panel. A secondary function can appear on one or more function panels. A function panel also can consist entirely of secondary functions. Use the Function Panel Editor to build function panels.

You also can add help for each control on a panel. Add help as you create each panel.

Writing the Function Code

After you name the function and define its parameter list, write the code to implement the function. LabWindows/CVI provides development tools for testing and debugging your code. The instrument driver you create uses full C language source code.

To develop the instrument driver source code, follow the guidelines in Chapter 3, *Programming Guidelines for Instrument Drivers*, of the *LabWindows/CVI Instrument Driver Developers Guide*.

Operating the Driver

After you create the .c (or .obj or .dll), .h, and .fp files, you can operate the instrument driver. To load the driver, right-click the Instruments folder in the Library Tree and select **Load Instrument**. Operate every function panel that you have created. Then, use the panels to generate a sample program to verify operation of the driver.

Testing the Instrument Driver

Before you distribute an instrument driver, you should fully test it. Test it from within the LabWindows/CVI interactive program and as a stand-alone application. A suggested testing sequence for instrument drivers follows.

⚠ **Caution**   Be sure to save copies of the original instrument source files in a separate directory.

1. Load the instrument driver and execute all functions from the function panels.
2. Verify correct operation of all functions.
3. Create and run a sample application program that exercises all the functions in the driver within LabWindows/CVI.
4. Verify correct operation of the application program.

Documenting the Driver

The final step in creating an instrument driver is to document the driver. The .doc file describes the purpose of the driver, the function tree, and function panels. The file also contains a function reference list that explains the syntax of each function in the driver. Chapter 3, *Programming Guidelines for Instrument Drivers*, of the *LabWindows/CVI Instrument Driver Developers Guide* contains guidelines and suggestions for documenting your instrument driver.

6

Using Function Panels

A function panel is a graphical representation of the functions in LabWindows/CVI libraries and instrument drivers. The controls on the function panels represent parameters. Some controls provide dialog boxes to help you select input for parameters. These controls have a **...** button next to them. You can use function panels to help generate and test function calls within LabWindows/CVI.

A function panel generates one or more function calls with the function parameters you specify in the function panel. LabWindows/CVI can execute these functions immediately in the Interactive Execution window. When you execute a function panel, LabWindows/CVI copies the generated code to the Interactive Execution window and executes it. The first time you execute a function panel for an instrument driver or library, LabWindows/CVI creates and executes a `#include` statement for the header file associated with the instrument driver or library.

Instead of executing the function call, you can choose to copy the function call code to a Source window. You can later recall the function panel from the Source window by selecting **View»Recall Function Panel**.

Normally, you use function panels to call into instrument drivers and libraries in the Library Tree. Also, you can use function panels to call functions in the project, as long as the functions are declared in the Interactive Execution window. Thus, you can create function panels for functions that you call frequently, even if you do not keep the functions in a separate file.

Accessing Function Panels

You can access a function panel for an instrument driver from the **Instrument** menu or the Instruments folder or for a library from the Library Tree or the **Library** menu. After you select an instrument or library name, choose a function panel by making a selection from the Select Function Panel dialog box or from the Library Tree.

Functions are organized in a multi-level structure called a function tree. The function tree groups functions into various classes according to the operation they perform. When the Select Function Panel dialog box contains class names, you can select a class name to view the next level of the function tree until you reach a list of function panels.

You also can move through the list of functions without using the tree structure. Use the **Flatten** or **Flatten Libraries** option to replace the function class hierarchy with a list of all

function panels at or below the current level. Once you select a function panel, the function panel commands **Previous Function Panel Window**, **Next Function Panel Window**, **First Function Panel Window**, and **Last Function Panel Window** give you access to function panels in this linear manner.

You can access function panels in other ways as well. For instance, you might want to return to a panel you recently used or recall a panel from the text of a function call in a Source window. The commands that give you access to panels in these and other ways are in the **View** menu of the Source window. A similar set of commands exist in the **View** menu of the Function Panel window. The Function Panel window is a collection of panels that represent all functions that users can interactively call from that window.

The following figure shows the Configure Channel Function Panel window for the IviScope Class Driver. The figure contains a function panel that corresponds to the IviScope_ConfigureChannel function. You can use controls on function panels to specify parameters for the functions. The generated code box at the bottom of the window displays the function calls these function panels generate.
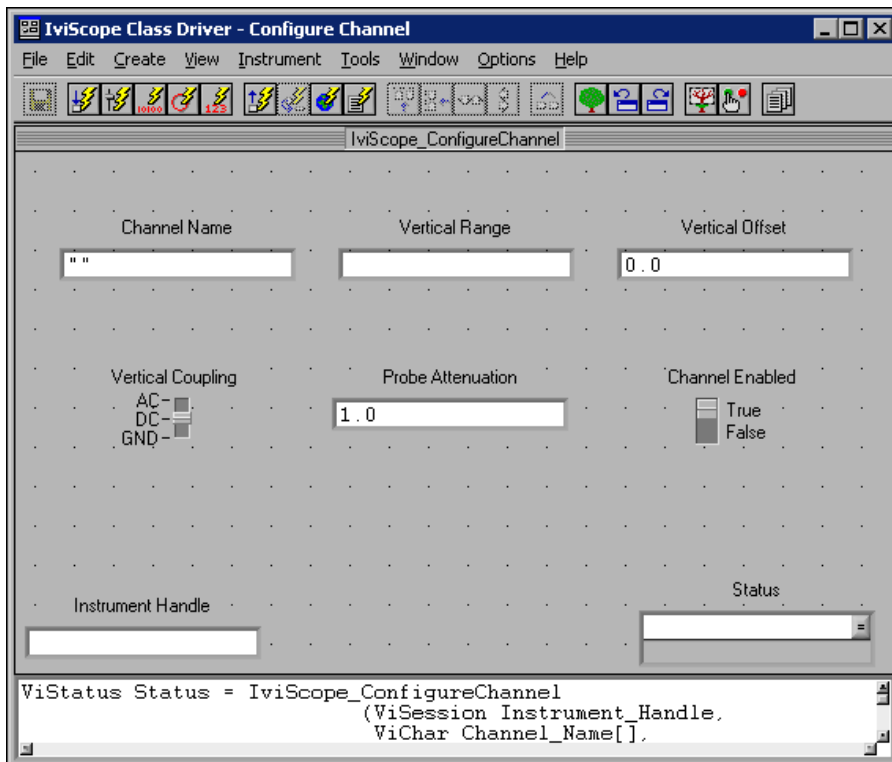
Multiple Function Panels in a Window

The Function Panel window can contain more than one function panel. Each function panel corresponds to one function, with the controls on that function panel manipulating the parameters to that function call. You can disable individual functions by selecting **Edit»Edit Function** and selecting **Function Disabled** from the dialog box. Disabled function calls do not appear in the generated code box, therefore, you cannot execute or insert them into a Source window. Each function panel occupies a position in the window. To modify the order of the function panel, select **Edit»Edit Function** and change the **Function Position** option.

✎ **Note**   In general, National Instruments recommends that you have only one function panel per window.

Generated Code Box

The generated code box at the bottom of the Function Panel window displays the code the function panels produce when you manipulate the panel controls. The generated code box displays up to three lines of code at a time and is scrollable.

Function Panel Controls

Function panel controls specify parameters in a function call.

✎ **Note**   Press <Page Up> or <Page Down> to move the input focus across multiple function panels in one window. Press <Ctrl-Page Up> and <Ctrl-Page Down> to move from one Function Panel window to the next.

The way you specify parameter values differs for each type of control.

Specifying Return Value Control Parameters

A return value control displays a value that a function returns as a return value rather than as a formal parameter.

For scalar return values, you can leave the control blank. LabWindows/CVI generates a temporary variable when you run the function panel.

If you type a variable name into a return control, you must define the variable statically in the Interactive Execution window or define it elsewhere and declare it as `extern` in the Interactive Execution window before you execute the function. To define a variable, select **Code»Declare Variable** in the Function Panel window. You can select **Code»Select Variable** to choose a variable or expression that you have used before. The type of value you

enter must agree with the data type of the control. To determine the data type of the control, press <F1> or right-click the control to view the help. After executing the function, the return value control displays the value for the variable beneath the variable name.

Specifying Input Control Parameters

An input control accepts a value you type in from the keyboard. An input control can have a default value associated with it. This value appears in the control when the panel first appears.

To specify a parameter for an input control, select the control and type a variable name, numeric value, or valid expression. Before you execute a Function Panel window, any names you type into input controls must be defined statically in the Interactive Execution window or defined elsewhere and declared as extern in the Interactive Execution window. To define a variable, select **Code»Declare Variable** in the Function Panel window. You can select **Code»Select Variable** to select a variable or expression that you have used before. The type of value you enter, whether it is a constant, expression, simple variable, or array, must agree with the data type of the control. To determine the data type of the control, press <F1> or right-click the control to view the Help window.

Specifying Numeric Control Parameters

A numeric control behaves like an input control except that it accepts only numeric values.

If you want to type a variable name into a numeric control, select **Options»Toggle Control Style**.

Specifying Slide Control Parameters

With a slide control, you select one item from a list of options. The position of the slider, the cross-bar on the slide control, determines the value LabWindows/CVI places in the function call.

As you move the slider on the control, the corresponding argument in the function call in the generated code box changes.

If you want to type a variable name into a slide control, select **Options»Toggle Control Style**.

Specifying Binary Control Parameters

This control is similar to the slide control, but the binary control has only two positions.

If you want to type a variable name into a binary control, select **Options»Toggle Control Style**.

Specifying Output Control Parameters

The output control displays a value that an executed function determines.

To specify a parameter for an output control, select the control and type the variable name. An output control parameter must be an array name or the address of a scalar or structure. For non-array parameters, you can leave an output control blank. LabWindows/CVI generates a temporary variable when you run the function panel. If the output control requires an array, or if you type a variable name into the output control, the variable must be defined statically in the Interactive Execution window or defined elsewhere and declared as extern in the Interactive Execution window before executing the function. To define a variable, select **Code»Declare Variable** in the Function Panel window. You can use **Code»Select Variable** to select a variable or expression that you have used before.

To view the value of an output control parameter after LabWindows/CVI executes the function, double-click the lower half of the output control to open the Variables window.

Using a Global Control

A global control displays the contents of global variables in a library function. You can use global controls to monitor global variables the function does not specifically return as results. Global controls are read-only controls. You cannot alter the content, and the controls do not contribute parameters to the generated code.

Common Control Function Panel

A Function Panel window can contain a special function panel called a *common control* function panel. The first *n* controls on a common control function panel specify the first *n* parameters of all functions in the Function Panel window. For more information about creating a common control function panel, refer to the *Create Menu for the Function Panel Editor* section of Chapter 9, *Function Panel Editor*.

Convenient Viewing of Function Panel Variables

Select **Code»View Variable Value** or **Add Watch Expression** to view the contents of arrays, structures, and global variables that exist in function panel controls. Depending on the type of the variable or expression, the Variables, Array Display, String Display, or Watch window appears with the variable or expression highlighted.

File Menu for Function Panel Windows

The commands in the **File** menu for Function Panel windows work in the same way as the commands in the **File** menu for the Workspace window. Refer to the *File Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*, for more information.

File»Add Program File to Project

The **Add Program File to Project** command adds the instrument driver program file associated with the `.fp` file of the current Function Panel window to the project list.

Code Menu for Function Panel Windows

This section contains detailed descriptions of the **Code** menu options for Function Panel windows.

Code»Run Function Panel

Use the **Run Function Panel** command to execute the code in the generated code box. When you select **Run Function Panel**, the following actions take place:

- LabWindows/CVI automatically inserts the header file for the library or instrument driver into the Interactive Execution window if it is not already there.

- LabWindows/CVI generates temporary variables for blank scalar output controls.

- LabWindows/CVI copies the generated function(s) to the Interactive Execution window.

- LabWindows/CVI executes the code. During execution, the **<<Running>>** menu appears in the upper left corner of the function panel menu bar.

- LabWindows/CVI displays the new values for output, return values, and global variable controls.

Code»Declare Variable

Use this command to declare a variable to be placed in the currently active control on the function panel.

- **Variable Type**—Indicates the data type associated with the currently active control on the panel. You can use more than one data type for some controls. In such cases, a ring control allows you to select the data type.

- **Variable Name**—Specifies the name of the variable you want to declare. LabWindows/CVI automatically prefixes scalar output variables with an ampersand (&).

- **Number of Elements**—Appears when the currently active control is for an array or a string. Enter the number of elements.

- **Execute declaration in Interactive Window**—Executes the variable declaration immediately in the Interactive Execution window.

- **Add declaration to top of target file (filename)**—Inserts a copy of the declaration at the top of the file you select using **Set Target File**.

- **Add declaration to current block in target file (filename)**—Inserts a copy of the declaration at the beginning of the code block that contains your current position.

- **Set Target File**—Sets the destination file for the **Insert Function Call** command. **Set Target File** opens a dialog box from which you can select any open Source window or the Interactive Execution window as the destination file.

- **OK**—Declares the variable according to the options you selected.

- **Cancel**—Cancels the operation and closes the Declare Variable dialog box.

When you use the **Declare Variable** command, LabWindows/CVI always declares the variable using the static storage class, unless you enable the **Add declaration to current block in target file** option.

In addition to generating the variable declaration, **Declare Variable** also places the variable name in the currently active control, overwriting the previous contents of the control.

If the currently active control already contains a syntactically correct variable name, it appears in the **Variable Name** text box when the Declare Variable dialog box first appears.

Code»Clear Interactive Declarations

Variables you declare in the Interactive Execution window remain in effect until you explicitly remove them. This feature lets you use these variables in succeeding executions of the Interactive Execution window and also enables different function panels to access the same variables.

The **Clear Interactive Declarations** command removes the variables without deleting the contents of the Interactive Execution window.

Code»Select Value

The **Select Value** command can help you select input for parameters. The **Select Value** command appears when a function panel control provides a dialog box from which you can make a selection. For example, if a control takes a filename as a value, **Select Value** opens a standard Open File dialog box from which you can browse to a file.

Controls with a **...** button next to them indicate that LabWindows/CVI provides a dialog box for input.

Code»Select UIR Constant

The **Select UIR Constant** command can help you use the function panels for the User Interface Library. Use this command to select from the list of constant names associated with the objects in your `.uir` files.

When you specify a parameter for an input control that can accept a panel resource ID, control ID, menu bar resource ID, menu ID, or menu item ID, use **Select UIR Constant** to open the Select UIR Constant dialog box.

The list in **User Interface Resource Files** contains all the `.uir` files open or in the project. Only constants from the currently selected `.uir` file appear in the list at the bottom of the dialog box. Click a file to select it.

Constant Type contains the categories of constant names to display in the list below **Constant Type**.

To copy the currently selected constant name into the function panel control, click **OK**.

Click **Cancel** to cancel the operation and close the Select UIR Constant dialog box.

Note   If you attempt to use **Select UIR Constant** on the **Panel Handle** and **Menu Bar Handle** parameters that appear on most User Interface Library function panels, an error message appears. These parameters take the values returned from `LoadPanel` and `LoadMenuBar`, so an attempt to select `.uir` constants fails.

You can use **Select UIR Constant** in user-defined panels. That way, the command is available to function panels for user libraries that you build on top of the User Interface Library.

Code»Select Attribute Constant

The **Select Attribute Constant** command replaces the **Select UIR Constant** command in the **Code** menu in function panels that set or get attribute values. The User Interface Library, the VISA Library, and IVI instrument drivers have such functions. Examples are `GetCtrlAttribute`, `SetCtrlAttribute`, `GetPanelAttribute`, and `SetPanelAttribute` in the User Interface Library. The panels for these functions each contain an Attribute ring control and a corresponding Value input control. When either of these two controls are active, the **Select Attribute Constant** command appears in the **Code** menu. The action of the command differs based on whether the Attribute or Value control is active.

Code»Select Variable or Expression

This command opens the Select Variable or Expression dialog box, which provides a list of previously used variables or expressions that have data types that are compatible with the currently active function panel control. LabWindows/CVI enables the command only when the currently active function panel control is one that accepts text entry. When you select a variable or expression from the list, LabWindows/CVI copies it into the function panel control. The **Select Variable** command can significantly reduce the amount of keyboard entry necessary when using function panels.

The Select Variable or Expression dialog box contains the following information:

- **Data Type of Control**—Indicates the data type of the currently active function panel control.

- **Variable or Expression**—Contains the variables and expressions that have data types compatible with the data type of the control.

- **Data Type**—Indicates the data type of each variable and expression.

- **Show Project Variables**—Adds to the list global variables, static and non-static, defined in project files that have been successfully compiled. If source code browse information is available, this option also adds to the list local variables that are in scope at the location of **Code»Insert Function Call**.

- **Build The Project**—Builds the project. **Show Project Variables** does not know about all of the variables in your project if you have added or removed variables from your project, built your project without source code browse information, or have never built your project before. You can select this option to build the project and update the list of variables and expressions.

- **OK**—Closes the dialog box and copies the variable or expression into the function panel control. LabWindows/CVI adds a leading ampersand (&) when the function panel control is an output control. LabWindows/CVI also adds one or more leading asterisks (*) or a trailing array indexation ([0]) when necessary to correctly match the data type of the control.

- **Cancel**—Cancels the operation and closes the Select Variable or Expression dialog box.

You can sort the entries in the Select Variable and Expression dialog box.

Sorted List Box Entries

LabWindows/CVI first sorts the entries in the list box by data type. The most compatible data types appear first. The exception is that some function panel controls use meta data types, such as `Numeric Array`, `Any Array`, or `Any Type`. Such controls are equally compatible with a wide range of data types. In this case, the order of data types does not indicate differing degrees of compatibility.

Within each data type, LabWindows/CVI sorts the entries alphabetically by the variable/expression text.

Included Variables and Expressions

The following items are included in the Select Variable or Expression dialog box.

- Variables you declare in the Interactive Execution window

- Variables you declare using the **Declare Variable** command in a function panel

- Variables or expressions used in function panels you execute

- Variables or expressions used in function panels from which you insert code into a Source window

- User Interface panel handle variables that CodeBuilder adds to a Source window

- Variables declared as global or static global in a project file that has been successfully compiled if you enable **Show Project Variables** in the dialog box

LabWindows/CVI removes some or all these items from memory when you unload or deactivate the current project or when you select **Build»Clear Interactive Declarations**.

Data Type Compatibility

Compatibility between data types is a more complex issue than you might expect. LabWindows/CVI uses a number of heuristics. The heuristics differ based on whether the variable is known to the compiler.

Variables known to the compiler include variables you declare in the Interactive Execution window and variables you declare in project files that you have successfully compiled. For such variables, LabWindows/CVI uses the following factors to determine whether the variable is type-compatible with a function panel control.

- LabWindows/CVI reduces data types you declare with the `typedef` keyword to their most intrinsic type, as long as the `typedef` is known to the compiler. For example, assume the compiler has processed the following declarations:

```
typedef int typeA;
typedef int typeB;
typedef typeB typeC;
```

   A variable of type `typeA` is an exact match for a function panel control that has type `typeC`.

- LabWindows/CVI considers all numeric types compatible with each other except that floating-point variables or expressions are not considered compatible with integer function panel controls.

- LabWindows/CVI considers types that have the same base type but differ in levels of indirection to be compatible. For example, the following types are all compatible:

```
int
int *
int **
int []
```

To be included in the Select Variable or Expression dialog box, an expression or a variable name that the compiler does not know must match exactly the data type of the function panel control. An example of a variable name not known to the compiler is one used in a function panel from which you insert code into a Source window.

> **Note** An expression or variable name that the compiler does not know can be associated with multiple data types. For instance, you might use the same variable name in an `int` control and a `double` control. If the variable is not known to the compiler, LabWindows/CVI has no way of knowing the true data type of the variable name. Thus, you might see the variable name associated with different data types.

Code»Insert Function Call

The **Insert Function Call** command copies the generated code to the selected window at the current location of the keyboard cursor. You can copy code to any open Source window or to the Interactive Execution window. To determine the destination window, select **Code»Set Target File**.

If the destination window contains selected text, LabWindows/CVI displays a dialog box that gives you the option of replacing the selected text or inserting the generated code after the selected text.

Code»Set Target File

Use the **Set Target File** command to set the destination file for the **Insert Function Call** command. **Set Target File** opens a dialog box from which you can select from a new Source window, any open Source window, or the Interactive Execution window.

Code»View Variable Value

Use the **View Variable Value** command to view the contents of arrays, structures, and global variables that appear in a function panel. Highlight the variable that you want to see and select **View Variable Value**. Depending on the type of the variable, the Variables, Array Display, or String Display window appears with the variable highlighted.

Code»Add Watch Expression

Use the **Add Watch Expression** command to view the value of an expression that appears in a function panel. Highlight the expression you want to see and select **Code»Add Watch Expression**. The Watch window appears with the expression highlighted.

View Menu for Function Panel Windows

This section contains detailed descriptions of the **View** menu options for Function Panel windows.

View»Toolbar

Use the **Toolbar** command to toggle between viewing or not viewing the Function Panel window toolbar.

View»Error

If an error occurs during the execution of a function panel, you can use the **Error** command to toggle between the error message and the code in the generated code box.

View»Include File

The **Include File** command displays the include file associated with the library or instrument driver in a Source window. The include file contains all the function prototypes for the library or instrument driver.

View»Current Tree

The **Current Tree** command displays the Select Function Panel dialog box for the most recently used function panel, making it easy for you to return to the location of the current panel in the function tree.

View»Function Panel History

The **Function Panel History** command displays a scrollable list of the function panels you have used during the current LabWindows/CVI session. You can display function panels from the list as new windows, or you can overwrite the current Function Panel window.

View»Find Function Panel

When you select **Find Function Panel**, a dialog box appears in which you can enter the name of a function. You can enter just a substring, and **Find Function Panel** finds all functions that contain that substring anywhere in their names. For instance, if you enter ctrl and click **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use regular expressions in your search string. Refer to the *Regular Expression Characters* section of Chapter 4, *Source and Interactive Execution Windows* for a list of regular expressions.

If a function panel exists for the function, LabWindows/CVI displays the panel. If two or more function panel windows exist for the function, LabWindows/CVI displays a list of the function panels.

The default shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

View»Previous Function Panel

The **Previous Function Panel** command displays the previous function panel in the current Function Panel window. This option applies only to multiple function panels in a window.

View»Next Function Panel

The **Next Function Panel** command displays the next function panel in the current Function Panel window. This option applies only to multiple function panels in a window.

View»Previous Function Panel Window

The **Previous Function Panel Window** command opens the Function Panel window that precedes the current Function Panel window in the same function free.

The rotation order for the function tree is circular. If the first Function Panel window in the tree is visible on the screen, selecting **Previous Function Panel Window** displays the last Function Panel window in the tree. If the last Function Panel window in the tree is visible, selecting **Next Function Panel Window** displays the first Function Panel window in the tree.

View»Next Function Panel Window

The **Next Function Panel Window** command opens the Function Panel window that follows the current Function Panel window in the same function tree.

View»First Function Panel Window

The **First Function Panel Window** command displays the first Function Panel window in the function tree.

View»Last Function Panel Window

The **Last Function Panel Window** command displays the last Function Panel window in the function tree.

Instrument Menu for Function Panel Windows

The **Instrument** menu for Function Panel windows works in the same way as the **Instrument** menu for the Workspace window. For more information about these commands, refer to the *Instrument Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Library Menu for Function Panel Windows

The **Library** menu for Function Panel windows works in the same way as the **Library** menu for the Workspace window. For more information about these commands, refer to the *Library Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Tools Menu for Function Panel Windows

The **Tools** menu for Function Panel windows works in the same way as the **Tools** menu for the Workspace window. For more information about these commands, refer to the *Tools Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Window Menu for Function Panel Windows

The **Window** menu for Function Panel windows works in the same way as the **Window** menu for the Workspace window. For more information about these commands, refer to the *Window Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Options Menu for Function Panel Windows

This section contains detailed descriptions of the **Options** menu items for Function Panel windows.

Options»Default Control

Default Control resets a control to its default value and configuration.

Options»Default All

Default All resets all the controls on the current Function Panel window to their default values and configurations.

Options»Toolbar

This option opens the Customize Toolbars dialog box. For more information about customizing the toolbar, refer to the *Toolbars in LabWindows/CVI* section of Chapter 2, *Workspace Window*.

Options»Exclude Function

The **Exclude Function** command disables the current function panel so that the function call does not appear in the generated code box and is not in effect when you select **Code»Run Function Panel** or **Insert Function Call**. This option is available only for multiple function panels in a window.

Options»Toggle Control Style

Slide, binary, and ring controls insert a number into a function call in the generated code box. The value of this number depends on the item you select in the control. You can override the configured values of these controls by using the **Toggle Control Style** command.

Toggle Control Style replaces a slide, binary, or ring control with an input control. You can use this input control to enter a variable name, constant, or expression. This entry appears in the generated code box in the same position as the parameter that the original control produced.

The variable name or constant that you enter must match the type specified for the control, such as short, long, single-precision, double-precision, string, and so on. Otherwise, a syntax error occurs when you execute the function.

Options»Change Format

Use **Change Format** to change the numeric format for scalar controls. The list of formats depends on the data type associated with the control.

You can display short and long data types in decimal, hexadecimal, octal, or ASCII form. You can display real numbers in floating-point or scientific format.

Options»Open Function Panels in New Window

Enable this command to open all function panels that you select in a new window. If you do not select this option, each function panel you select replaces the current function panel.

Options»Go to Source After Inserting Code

Enable this command to move the cursor to the source file after you insert a function from a function panel.

Options»Edit Function Panel

The **Edit Function Panel** command puts the Function Panel window in edit mode.

✎ **Note** You cannot edit the function panels of the LabWindows/CVI libraries or user libraries.

Help Menu for Function Panel Windows

Use the commands in the **Help** menu to access information about LabWindows/CVI. These commands work in the same way as the **Help** menu commands for the Workspace window. Refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Help»Control

The **Control** command displays help about the currently highlighted control. To select control help with the mouse, right-click the control you want help about.

Help»Function

The **Function** command displays help about the function that the current function panel generates. To select function help with the mouse, right-click the function panel.

Help»Online Function Help

The **Online Function Help** command opens the *LabWindows/CVI Help* to the help topic for the current function panel.

7

Function Tree Editor

About the Function Tree and Function Tree Editor

The function tree defines the hierarchical structure in which functions of an instrument driver are grouped. Use the Function Tree Editor to create and modify the function tree for an instrument driver.

To invoke the Function Tree Editor, select **File»New»Function Tree (*.fp)** or **File»Open»Function Tree (*.fp)**. You also can drop a .fp file into the Window Confinement Region of the Workspace window.

When you invoke the Function Tree Editor, a new Function Tree Editor window appears. If you select **New** to create a new function tree, you see a blank Function Tree Editor. Right-click the background or select **Create»Instrument** to create an instrument.

If you select **Open** to edit an existing function tree, the function tree for the file you selected appears in the window. To edit a function panel, double-click the function in the function tree.

File Menu for the Function Tree Editor

The commands in the **File** menu for Function Tree Editor work in the same way as the commands in the **File** menu for the Workspace window. Refer to the *File Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*, for more information.

File»Add Program File to Project

The **Add Program File to Project** command adds the instrument driver program file associated with the .fp file of the current Function Panel window to the project list.

Edit Menu for the Function Tree Editor

Use the commands in the **Edit** menu to edit the entries in the function tree.

✎ **Note** When you cut, copy, or paste a class, all of its subclasses and functions are cut, copied, or pasted as well. Similarly, when you delete a class, all of its subclasses and functions are deleted.

Edit»Cut

Cut deletes the selected function or class from the tree and copies it to the clipboard.

Edit»Copy

Copy copies the selected function or class from the tree to the clipboard.

Edit»Paste Above

Paste Above inserts the contents of the clipboard into the tree above the selected node.

Edit»Paste Below

Paste Below inserts the contents of the clipboard into the tree below the selected node.

Edit»Delete

Use the **Delete** command to remove the selected node from the function tree.

Edit»Edit Node

Edit Node lets you edit the instrument, function, or class name on the highlighted line.

Edit»Edit Help

Edit Help lets you add context-sensitive help information to the function tree.

Edit»Edit Function Panel Window

Edit»Edit Function Panel Window lets you edit the selected Function Panel window in the Function Panel Editor. The Function Panel window is a collection of panels that represent all functions that users can interactively call from that window.

Edit».FP Auto-Load List

Use the **.FP Auto-Load List** command to specify other instrument drivers that the current instrument driver depends on. LabWindows/CVI loads these instrument drivers automatically when you load the current instrument driver.

The **.FP Auto-Load List** command opens a dialog box in which you can list simple `.fp` filenames. Do not include drive or directory names. When you load the current instrument driver, LabWindows/CVI also tries to load the instrument drivers identified by these `.fp` filenames.

LabWindows/CVI looks for these `.fp` files in the following sequence:

1. If the `.fp` file is under the IVI framework directory, LabWindows/CVI looks for the `.fp` file using the following pathname, where *IVIfrmwk* is the IVI framework directory and `prefix` is the instrument prefix:

    IVIfrmwk\drivers*prefix**prefix*.fp

2. If the `.fp` file is under the VXI*plug&play* framework directory, LabWindows/CVI looks for the `.fp` file using the following pathnames, where *vppfrmwk* is the VXI*plug&play* framework directory and *prefix* is the instrument prefix:

    vppfrmwk\support*prefix**prefix*.fp
    vppfrmwk*prefix**prefix*.fp

3. LabWindows/CVI looks in the directory of the referencing `.fp` file.

4. LabWindows/CVI looks for the `.fp` files in the instrument directories list. Edit the instrument directories list by selecting **Instrument»Search Directories**.

5. LabWindows/CVI looks for the files in the `instr` directory under the directory where LabWindows/CVI is installed.

If LabWindows/CVI cannot find the `.fp` file, you can look for the file using a file dialog box. If you find the `.fp` file, LabWindows/CVI prompts you to add the directory to the instrument directories list and prompts you to add the file to the project.

If an auto-loaded `.fp` file has no classes or function panels, it does not appear in the **Instrument** menu. This is useful for support modules that contain no user-callable functions.

When you select **Instrument»Unload**, all auto-loaded `.fp` files appear in the Unload Instrument dialog box. Auto-loaded instruments are not unloaded automatically when the dependent instrument is unloaded.

Edit»Find

Use the **Find** command to locate a particular text string in the function panel file. You can search for text in node names; function names; control labels; control values; item labels in ring, slide, and binary controls; message control text; and help text. When you search in help text, you cannot search in any of the other items at the same time. The search begins at the node that is currently selected.

If the **Find** command opens the Help Editor and you do not use the button bar, you must return to the Function Tree Editor to continue searching throughout the panel. The **Find** command in the Help Editor searches only within the window. On the other hand, the **Find** command in the Function Panel Editor continues searching through the entire function panel file.

Edit»Replace

The **Replace** command works in the same way as the **Find** command except that you can replace the search string with another search string.

Create Menu for the Function Tree Editor

Use the commands in the **Create** menu to create a new function tree or add new functions and classes to an existing function tree.

Create»Instrument

Use the **Create»Instrument** command to create a new function tree. When you select **Instrument**, the Create Instrument Node dialog box appears. Enter the following information:

- The name of the instrument (up to 40 characters).

- The prefix that you want LabWindows/CVI to add to the beginning of each function name. The prefix cannot exceed eight characters, if you are using 5.0.1 or earlier as the default format; the prefix can use up to 31 characters for instrument drivers using a 5.5 and later as the default format. Do not include the underscore (_) separator in your prefix. LabWindows/CVI adds an underscore (_) separator to the prefix before appending the function name to it.

- The default qualifier that provides information to LabWindows/CVI about access options and conventions. LabWindows/CVI uses this qualifier for all functions in the instrument driver unless a function specifies a different qualifier. This option appears only if you select **CVI 5.5 and later** as the **Default Format for New Files** in the **FP File Format** dialog box.

  Example: Many instrument drivers use the `_VI_FUNC` macro as the function qualifier. The `_VI_FUNC` macro is defined in the `visatype.h` include file. The definition of the `_VI_FUNC` macro varies based on the target operating system and application development environment.

The `Create Class or Function Panel Window` line appears beneath the instrument name. Add functions and classes to the function tree using the **Create»Function** and **Class** commands.

Create»Class

Use the **Class** command to add a new class to a function tree.

When you select the **Class** command, the Create Class Node dialog box appears. Enter the name that you want to appear in the Select Function Panel dialog box that opens when the user selects the instrument from the **Instrument** menu.

In the function panel hierarchy, you can insert up to eight levels of classes.

📝   **Note**   A function tree can contain a combination of up to 32,000 functions and classes.

Create»Function Panel Window

Select **Create»Function Panel Window** to add a new function to a function tree. The Function Panel window is a collection of panels that represents all functions that users can interactively call from that window.

When you select the **Function Panel Window** command, the Create Function Panel Window Node dialog box appears. Enter the following information:

1. In **Name**, enter the name that you want to appear in the Library Tree or the Function Panel Selection dialog box.

2. In **Function Name**, enter the actual code name used in the instrument driver for the function being added. This function name must be valid for the current language.

    The name of every function in an instrument driver begins with a common prefix taken from the name of the instrument driver. Do not enter the prefix of the function name in the Create Function Panel Window Node dialog box. LabWindows/CVI automatically adds the prefix to each function name. To change the prefix, select **Edit»Node** when you select the instrument node.

3. In **Qualifier**, enter the default qualifier that provides information to LabWindows/CVI about access options and conventions. This qualifier overrides the default qualifier specified for the instrument. This option is available only if you select **CVI 5.5 and later** as the **Default Format for New Files** in the FP File Format dialog box.

    Example: Many instrument drivers use the _VI_FUNC macro as the function qualifier. The _VI_FUNC macro is defined in the visatype.h include file. The definition of the _VI_FUNC macro varies based on the target operating system and application development environment.

Adding a Function to an Empty Tree or Class

Complete the following steps to add a function to an empty tree or class.

1. Select the line that contains `Create Class or Function Panel Window`.

2. Select **Create»Function Panel Window**. The Create Function Panel Window Node dialog box appears.

3. Complete the Create Function Panel Window Node dialog box. The new function name appears in place of the `Create Class or Function Panel Window` message on the selected line.

Inserting a Function into an Existing Tree

Complete the following steps to insert a function at any level in an existing function tree.

1. Select an existing function or class at the level you want to place the new function.

2. Select **Create»Function Panel Window**. The Create Function Panel Window Node dialog box appears.

3. Complete the Create Function Panel Window Node dialog box. The new function is inserted on the line below the existing function or class. The function exists at the same level in the tree as the function or class that originally occupied the line.

Instrument Menu for the Function Tree Editor

The **Instrument** menu for the Function Tree Editor works in the same way as the **Instrument** menu for the Workspace window. For more information about these commands, refer to the *Instrument Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Library Menu for the Function Tree Editor

The **Library** menu for the Function Tree Editor works in the same way as the **Library** menu for the Workspace window. For more information about these commands, refer to the *Library Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Tools Menu for the Function Tree Editor

Many of the commands in the **Tools** menu for the Function Tree Editor work in the same way as the **Tools** menu for the Workspace window. For more information about these commands, refer to the *Window Menu for the Source and Interactive Execution Windows* section of Chapter 4, *Source and Interactive Execution Windows*.

Use the commands in the **Tools** menu to generate function definitions and declarations into your source and include files, jump to function definitions and declarations, generate .hpp files for the function tree, and invoke the Create IVI Instrument Driver wizard and attribute editor.

Tools»Generate C++ Wrapper

Generate C++ Wrapper generates a .hpp file for the function tree. This file contains the definition of a C++ class for the instrument driver. LabWindows/CVI generates a wrapper function for each function in the function tree. Required C++ member functions, such as constructors and destructors, also are generated. The generated class references classes that are defined in ivibase.hpp and iviexcpt.hpp, which are located in the cvi70\include

directory. You can edit these classes to customize the generated wrapper class. This command is available only for IVI drivers.

Tools»Enable Auto Replace

Enable Auto Replace enables the updating of instrument driver source files to reflect changes to function names in the function tree. This option is global to LabWindows/CVI, so enabling it in one Function Tree Editor window enables it for all Function Tree Editor windows. This command is dimmed for function trees that have not yet been saved.

When you enable this option, LabWindows/CVI updates the instrument driver `.c`, `.h`, and `.sub` files to reflect changes you make to function names or the instrument prefix in the Function Tree Editor window or Function Panel Editor window. When you change a function name, LabWindows/CVI prompts you for permission to update your instrument driver to reflect the new name. When you change the instrument prefix, LabWindows/CVI prompts you for permission to update your instrument driver to reflect the new prefix.

Tools»Customize Function Panels

Use the **Customize Function Panels** option to provide input selections for controls in function panels.

An example of input selections is the Select Attribute Constant dialog box in Get/Set...Attribute function panels. You can add similar dialog boxes for your function panels. For example, if one of your parameters requires the device number of a DAQ device, you can provide a customized dialog box with selections to help the user select input for the parameter.

To include input selections for parameters, you must associate a control in a function panel to a callback function in a DLL. You can make these associations in the Customize Function Panels and Customize Controls dialog boxes.

In the Customize Function Panels dialog box, **Parameters** lists input, numeric, and ring controls from function panels in the current instrument. The **Functions** column lists the function panels that contain the selected control. To open the Customize Controls dialog box, select one or more functions to customize and click the **Customize** button.

In the Customize Controls dialog box, you can select a DLL and callback function to attach to the parameter. LabWindows/CVI populates **Callback** with callback functions that the DLL exports. To remove a DLL and callback associated with a parameter, click **Clear Entry**.

Clicking **OK** in the Customize Controls dialog box populates the **Callback**, **DLL**, and **DLL path** columns in the Customize Function Panels dialog box with the callback function, DLL, and DLL path that you selected.

You also can access the Customize Controls dialog box by right-clicking a control when you edit a Function Panel window.

When you customize a function panel, LabWindows/CVI creates a function panel customization (`.fpc`) file. The `.fpc` file must be located in the same folder as the `.fp` file.

After you customize a function panel, LabWindows/CVI calls the callback you assigned when you press <Enter> in the customized control or when you click the **...** button next to the control.

LabWindows/CVI searches in the following locations to load the customization DLLs.

•   The DLL path specified in the Customize Controls dialog box

•   The folder where the function panel is located

•   The default system search path

Tools»Generate New Source For Function Panel

Generate New Source For Function Panel generates function definitions and declarations in your driver source and header files. If a function definition already exists, LabWindows/CVI prompts you for permission to update it. You can replace, insert above or below, or skip without updating. Your choice also applies to the declaration.

Tools»Go to Definition

Go to Definition opens the driver source file and jumps to the definition of the function that is currently selected in the function tree.

Tools»Go to Declaration

Go to Declaration opens the driver include file and jumps to the declaration of the function that is currently selected in the function tree.

Window Menu for the Function Tree Editor

The **Window** menu for the Function Tree Editor works in the same way as the **Window** menu for the User Interface Editor. For more information about these commands, refer to the *Window Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*.

Options Menu for the Function Tree Editor

Use the commands in the **Options** menu to select the help style, generate function prototypes, generate a `.doc` file, create a DLL project, select whether to enable IVI or VXI*plug&play* style, and save the function panel as an XML file.

Options»FP File Format

Use **FP File Format** to select the format for the current function panel file and the default format for new function panel files. Two function panel file formats are available. The following table lists the features of the two formats.

Table 7-1. .FP File Format Features

Feature	LabWindows/CVI 5.0.1 and Earlier Format	LabWindows/CVI 5.5 and Later Format
Maximum Instrument Prefix Length	8	31
Maximum Function Name Length	31	79
Maximum Class Name Length	31	79
Maximum Window Name Length	31	79
Maximum Type String Length	50	80
Default Text for Output Controls	Not Supported	Supported
Function Qualifiers	Not Supported	Supported
Set Precision on Numeric Controls	Not Supported	Supported

Options»Help Style

Use **Help Style** to choose the help style—**New (Recommended)** or **Old (LabWindows DOS)**—to use when you edit context-sensitive help for the function tree.

The new and old help styles differ significantly. The old help style maintains compatibility with function panels created in LabWindows version 2.3 or earlier. This help style uses the DOS/IBM character set so that it can display special extended ASCII characters that many older instrument drivers use. Also, the old style provides help for the entire Function Panel window, not the individual function panels within a Function Panel window.

The new help style uses the standard Windows character set and provides help for each individual function panel. In addition, the new help style automatically generates control name and data type information when displaying control help and automatically generates a function prototype when displaying function help. Also, the help text editor for the new style help uses word-wrap mode.

Changing the help style changes only how the program interprets help. If you use special extended ASCII characters in your help and then change to the new style, you must change the help text to a Windows-compatible character set.

Use the new help style whenever possible.

Options»Transfer Window Help to Function Help

Transfer Window Help to Function Help helps you convert function panels from old to new style. For each Function Panel window, the window help text is transferred to the first function unless the function already has help text.

Options»Generate Function Prototypes

Generate Function Prototypes creates an untitled `.h` file that contains prototypes for the functions in the function tree.

Options»Generate Documentation

Generate Documentation creates a window that contains a `.doc` file for the function panel file.

Options»Generate Windows Help

Generate Windows Help creates a Windows help file (`.hlp`), help project file (`.hpj`), and two source files (`.rtf` and `.whh`) that you can use with Microsoft Windows Help Compiler to create a Windows help file. You can choose the output language as either C or Visual Basic.

Options»Generate ODL File

Generate ODL File creates an Object Description Language (`.odl`) file for the instrument driver. The `.odl` file can be input to the MIDL program that comes with the Platform SDK. This is useful when you create a DLL version of the instrument driver. The MIDL program creates a type library that describes the function entry points in the DLL. Refer to MSDN for information about using this program.

Options»Generate DEF File

Generate DEF File generates a `.def` file for the instrument driver. External compilers use the `.def` file to compile instrument drivers into a DLL. The file contains entries to export each function in the function tree.

Options»Create DLL Project

The **Create DLL Project** command creates a LabWindows/CVI project (`.prj`) file, from the program file associated with the function panel file, that you can use to create a DLL. When you execute this command, LabWindows/CVI prompts you to enter a pathname for the project file. After the file is written, LabWindows/CVI prompts you about loading the project immediately. If you choose to load the file, your current project is unloaded.

Options»IVI/VXIplug&play Style

This option affects the contents of the DLL project that you create using the **Options»Create DLL Project** command. If you enable the **IVI/VXIplug&play Style** command, **Create DLL Project** adds project settings that allow the DLL, import libraries, and distribution kit you create to conform to various aspects of the IVI or VXI*plug&play* specification. You can modify all of these settings using commands in the **Build** menu of the Workspace window. The following list describes the default settings.

In the Target Settings dialog box for DLLs, the following options apply:

- The run-time support is set to **Instrument Driver Only**.

- _32 is appended to the base filename of the DLL but not to the base filename of the import libraries.

- In the DLL Import Library Choices dialog box, the **Generate import library for both compilers** option is enabled.

- In the Type Library dialog box, the following options apply:

  – The **Add type library resource to DLL** option is enabled.

  – The **Include links to help file** option is enabled.

  – **Function panel file** is set to the full pathname of the .fp file of the current Function Tree Editor window.

- In the **Change** option in the **Exports** section, the following options apply:

  – The **Export What** option is set to **Include File Symbols**.

  – The **Which Project Include Files** list contains the name of the include file associated with the .fp file of the current Function Tree Editor window.

In the Create Distribution Kit dialog box, the following options apply:

- The **Run-Time Engine Support** option is set to **Instrument Driver Only**. If you need the LabWindows/CVI Run-time Engine for the soft front panel executable, you must change this option to **Full Run-Time Engine** manually.

- LabWindows/CVI creates file groups that contain all the files that are required of a VXI*plug&play* or IVI instrument driver installation. For example, the import libraries for Visual C/C++ and Borland C/C++ are included, and their directory names are msc and bc for VXI*plug&play* and IVI. Files that you must create independently also are named in the file groups, even if they do not currently exist. These include the following files:

  – A Visual Basic include file, which you can create by selecting **Options»Generate Visual Basic Include** in the Source window (this file is not required for IVI drivers)

  – A documentation file, which you can create by selecting **Options»Generate Documentation** (this file is not required for IVI drivers)

  – A help file, which you can create by selecting **Options»Generate Windows Help** and the Windows Help Compiler (this file is not required for IVI drivers)

      –   A knowledge base file as defined in the VXI*plug&play* specification (this file is not required for IVI drivers)

      –   Files for a soft front panel executable (an empty file group is created for this)

In the Advanced Distribution Kit Options dialog box, the following options apply:

- **Executable Filename** is left empty. After you create a soft front panel executable and add it to the soft front panel file group, click on the **Select** button to specify the soft front panel executable as the **Executable Filename**.

- The **Installation Titles** names are set to `<instrument prefix> Instrument Driver`.

Options»Save in XML Format

The **Save in XML Format** command saves the current function panel in an XML format. When you select this option, a dialog box appears prompting you to enter the pathname under which to save the XML file. NI recommends using the extension `.fpx` for these files. Do not use the `.fp` extension. A document type definition (DTD) for this XML format is available in the `bin` directory. This file is named `cviFPX.dtd`.

Note   You can open the resulting `.fpx` file in any XML editor, text editor, or in Microsoft Internet Explorer. However, you must use the **Load from XML Format** option to import the file back into LabWindows/CVI as a function panel.

Note   If you have a large number of panels in your function panel, saving and loading a `.fpx` file can take significantly longer than saving and loading a comparable `.fp` file.

Options»Load from XML Format

The **Load from XML Format** command creates a new Function Panel window containing the function definitions generated by the **Save in XML Format** command. When you select **Options»Load from XML Format**, a dialog box appears prompting you for the pathname to the file. XML format function panel files usually have a `.fpx` file extension. A document type definition (DTD) for this XML format is available in the `bin` directory. This file is named `cviFPX.dtd`.

If the `.fpx` you are importing has a matching `.fp` already in the current workspace or is loaded as an instrument, the following dialog box appears.

LabWindows/CVI Message

The XML FP you are trying to open is either already in the current workspace or is loaded as an instrument. Select one of the following options: display the existing FP, replace the existing FP in memory, or cancel the load.

Display   Replace   Cancel
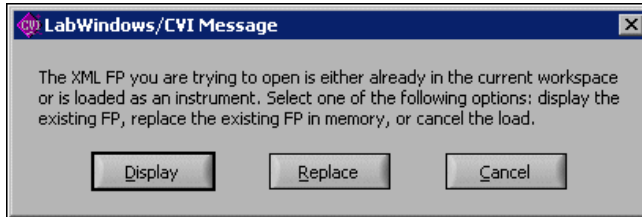
You have several options. You can choose to display the existing function panel in the function tree editor, replace the function panel that is already in memory with the one that you are importing, or cancel the load operation.

> **Note** In all cases, the function panel that you import is loaded only into memory and is not saved to disk until you explicitly choose the **Save** option.

> **Note** If you have a large number of panels in your function panel, saving and loading a `.fpx` file can take significantly longer than saving and loading a comparable `.fp` file.

Help Menu for the Function Tree Editor

Use the commands in the **Help** menu to access information about LabWindows/CVI. These commands work in the same way as the **Help** menu commands for the Workspace window. Refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Creating a Function Tree with Multiple Classes

In this example, you create a function tree with nested classes. Before beginning, select **File»New»Function Tree (*.fp)** to invoke the Function Tree Editor.

Complete the following steps to create a new instrument and function tree.

1.   Select **Create»Instrument**.

2.   Enter `Function Tree Examples` as the **Name** and `tree` as the **Prefix**. Click **OK**.

3.   Select **Create»Function Panel Window**.

4.   Enter `Function 1` as the **Name** and `fun1` as the **Function Name**. Click **OK**.

5.   Select **Create»Class**.

6.   Enter `Class 1` as the **Name**. Click **OK**.

7.   Select **Create»Function Panel Window**.

8.   Enter `Function 2` as the **Name** and `fun2` as the **Function Name**. Click **OK**.

9. Select **File»Save .FP File As** and save the file as `multclass`.

10. To view the structure of the function tree as it is seen by the instrument driver user, add the instrument to the project by selecting **File»Add .FP File to Project**. Double-click `multclass.fp` in the Project Tree in the Workspace window.

Moving and Copying Function Panels

You can move a function and its associated function panel in a function tree to a new position within the function tree. To move a function, drag the function and drop it into a new position. You also can cut a function and paste it into a new position.

Complete the following steps to cut and paste a function within a function tree.

1. Open `multclass.fp`.

2. Select `Function 1`.

3. Select **Edit»Cut**. The function disappears from the tree and is stored on the clipboard.

4. Select `Function 2`.

5. Select **Edit»Paste Above**. The function now appears under `Class 1`.

Instead of moving a function, you can replicate it. Complete the following steps to copy a function within a tree.

1. Select `Function 1`.

2. Select **Edit»Copy**.

3. Select `Class 1`.

4. Select **Edit»Paste Below**. The name `Function 1` now appears at the top of the tree and under `Class 1`.

📝 **Note**   Pasting functions and classes within the Function Tree Editor copies all items associated with the function or class, including controls and function panel help.

Using Existing Function Panels in a New Instrument Driver

Complete the following steps to copy function panels from an existing instrument driver to a new instrument driver.

1. Select **File»New»Function Tree (*.fp)**. A new blank Function Tree Editor appears on the screen.

2. Select **Create»Instrument**.

3. Name the instrument `New Instrument` and type `new` in **Prefix**. Click **OK**.

4. Select **Window»Function Tree** and select the file called `multclass`.

5. Select `Class 1`.

6. Select **Edit»Copy**.

7. Return to the `New Instrument` file through the **Window** menu.

8. Select **Edit»Paste Below**. `Class 1` and its associated functions appear in the new tree.

When you paste a class into a new tree, all information associated with the class and the functions of the class is retained.

Editing Items in the Function Tree

In this example, you edit the names displayed in the function tree. To edit all the function tree items, select **Edit»Edit Node**.

Complete the following steps to change the name of the instrument driver and its prefix.

1. Select `Function Tree Examples` from the function tree you created.

2. Select **Edit»Edit Node**. The Edit Instrument Node dialog box originally used to create the instrument appears.

3. Change the name of the instrument to `Tree #2` and the prefix to `tree2`. Click **OK**.

The changes in the instrument driver name appear at the top of the function tree in the Function Tree Editor. If you save the file, these changes also appear in the Library Tree. The changes to the prefix are reflected in the generated code window in each function panel.

8

Function Panel Editor

This chapter describes how to create and modify instrument driver function panels using the Function Panel Editor.

To create a new function panel, you must invoke the Function Panel Editor.

The following items appear on the function panel:

- The Instrument Name and Function Panel Name appear in the title bar of the Function Panel window.

- The Function Panel Editor menu bar appears at the top of the screen above the function panel.

- If you have enabled the toolbar, the toolbar appears under the menu bar.

- The Function Name appears in the title bar of the function panel.

- The Function Name appears with an empty argument list in the Generated Code window, below the Function Panel Editor window.

- By default, grid lines are enabled for the Function Panel Editor. Use grid lines to align controls. If you want to disable grid lines, select **Options»Grid Line Options**.

Invoking the Function Panel Editor

You can invoke the Function Panel Editor from the Function Tree Editor or from a function panel.

Invoking from the Function Tree Editor

To invoke the Function Panel Editor from the Function Tree Editor, complete the following steps.

1. Highlight the function that corresponds to the function panel you want to edit.

2. Select **Edit»Edit Function Panel Window** in the Function Tree Editor.

You also can invoke the Function Panel Editor by double-clicking the function name.

Invoking from a Function Panel

To edit a function panel that you are currently operating, select **Options»Edit Function Panel** in the Function Panel window. If the current function panel is for a library that is in the Library Tree, you cannot use the **Edit Function Panel** command.

File Menu for the Function Panel Editor

The commands in the **File** menu for Function Panel Editor work in the same way as the commands in the **File** menu for the Workspace window. Refer to the *File Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*, for more information.

File»Add Program File to Project

The **Add Program File to Project** command adds the instrument driver program file associated with the .fp file of the current Function Panel window to the project list.

Edit Menu for the Function Panel Editor

Many of the commands in the **Edit** menu for the Function Panel Editor work in the same way as the commands in the **Edit** menu for the User Interface Editor. For more information, refer to the *Edit Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*.

Edit»Undo and Redo

The **Undo** command reverses your last action. Use **Undo** to reverse the following actions:

- Move a control.
- Cut, copy, or paste a control.
- Make changes using the **Control Coordinates** command.

You lose undo information when you perform any of the following actions:

- Create a control.
- Edit a control, make a change, and click the **OK** button.
- Cut the panel.
- Close the window.
- Advance to another function panel in the same window.

The **Redo** command reverses the last **Undo** command.

Edit»Cut Controls and Copy Controls

To cut or copy controls to the clipboard, select the control you want to place on the clipboard and then select **Edit»Cut Controls** or **Copy Controls**. LabWindows/CVI places the selected control and its associated help on the clipboard. If you used the **Copy Controls** command, the control remains in the window. Use the **Cut Controls** command to delete controls from the window. Controls you cut or copy do not accumulate on the clipboard. LabWindows/CVI replaces a control every time you cut or copy a control to the clipboard.

To use the **Cut Controls** or **Copy Controls** commands, follow these steps:

1. Select the control you want to place on the clipboard by clicking the control or pressing <Tab> until the control is highlighted. Select multiple controls by dragging the mouse over the controls. You also can press <Shift-Click> to select multiple controls.

2. Select **Edit»Cut Controls** or **Copy Controls**.

Edit»Paste

The **Paste** command copies objects from the clipboard and places them on a Function Panel window. You can paste the same object as many times as you need to.

Controls or panels remain on the clipboard until you use **Cut Controls**, **Cut Panel**, **Copy Controls**, or **Copy Panel** again. The **New** and **Open** commands do not erase the clipboard.

You cannot paste a return value control on a function panel that already contains a return value control. A function panel can contain only one return value control.

Edit»Edit Control

You can modify an existing control with **Edit Control**. When you select **Edit Control**, you see the same series of dialog boxes you use to create the control. Refer to the *Create Menu for the Function Panel Editor* section for more information about controls.

Edit»Change Control Type

Use this command to change the type of a control. When you select **Change Control Type**, a dialog box appears that lists the available control types.

Select a control type from the dialog box. When you select a new control type, you see the same series of dialog boxes that you use to create the control.

If you change a control type from slide to ring, or vice versa, the new control type retains the option list associated with the old control.

Edit»Customize Controls

Refer to the *Tools Menu for the Function Tree Editor* section of Chapter 7, *Function Tree Editor*, for information about this command.

Edit»Edit Function

Use this command to modify an existing function panel. Selecting this command opens the Edit Function Panel dialog box, in which you can modify the function name and qualifier. You also can specify the function position, if you have multiple function panels, and whether or not to disable the function.

Edit»Control Coordinates

Use the **Control Coordinates** command to view and set the top and left coordinates and width of the controls on the panel. You can use the Control Coordinates dialog box to align and move controls.

Control Name displays a list of the controls in the function panel, along with the top and left coordinates and the width of each control.

Use the commands and ring controls to the right of **Control Name** to edit the top coordinate, left coordinate, and width of each control.

Top, **Left**, and **Width** contain default values until you highlight a specific control name and click **Extract**. When you click **Extract**, LabWindows/CVI displays the coordinates for the highlighted control in the ring controls.

You can change the top and left coordinates and width of a control by highlighting a control name in the list box, entering a value in the ring controls, and clicking **Apply**.

To align or move controls, complete the following steps:

1. Click the column to the left of a control name, and click **Extract** to copy the existing value of the control to the **Top**, **Left**, or **Width** ring controls.

2. To apply values to other control names, place a check next to the control names whose values you want to change. Click **Apply** to copy the values.

Edit»Find

Use the **Find** command to locate a particular text string in the function panel file. You can search for text in node names; function names; control labels; control values; item labels in ring, slide, and binary controls; message control text; and help text. You also can specify that the search be case-sensitive and whole word, use regular expressions, and wrap. When you search in help text, you cannot search in any of the other items at the same time.

The search begins at the function tree node for the current Function Panel Editor. The **Find** command always searches all controls on the panel regardless of whether any are currently selected.

If the **Find** command opens the Help Editor and you do not use the button bar, you must return to the Function Panel Editor or Function Tree Editor to continue searching throughout the function panel file. The **Find** command in the Help Editor window searches only within the window.

Edit»Replace

The **Replace** command works in the same way as the **Find** command except that you can replace the search string with another string.

Edit»Control Help

The **Control Help** command opens the Help Editor, in which you can add or modify context-sensitive help for a control.

Edit»Function Help or Window Help

This command opens the Help Editor, in which you can add or modify context-sensitive help for the entire function panel. **Function Help** corresponds to new style help and **Window Help** corresponds to old style help.

Create Menu for the Function Panel Editor

Use the **Create** menu to add controls, function panels, or a common control panel to a Function Panel window. You can create the following types of controls from the **Create** menu: input, slide, binary, ring, numeric, output, return value, global variable, and message.

Create»Input

An input control accepts a variable name or value entered from the keyboard. When you select **Create»Input**, the Create Input Control dialog box appears.

The following items appear in the dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Parameter Position**—The location of the control value in the function parameter list. For a control in a common control function panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a Function Panel window. The first position is 1.

For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control function panel. The first position after the controls in the common control function panel is 1. If there is no common control function panel, the first position is 1.

- **Data Type**—The data type of the item entered in the input control.

- **Default Value**—The default for the input control. The default must be a valid value, a constant name, or any other valid C expression.

- **Control Width**—The width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Slide

A slide control looks like a mechanical slide switch. A slide control specifies a parameter value depending upon the position of the cross-bar of the slide control. When you select **Create»Slide**, the Create Slide Control dialog box appears.

The following items appear in the dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Parameter Position**—The location of the control value in the function parameter list. For a control in a common control function panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a Function Panel window. The first position is 1.

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control function panel. The first position after the controls in the common control function panel is 1. If there is no common control function panel, the first position is 1.

- **Data Type**—The data type of the values in the slide control.

- **Default Value**—The default for the slide control. The default must be one of the labels specified in the Edit Label/Value Pairs dialog box.

- When you click the Edit Label/Value Pairs button, the Edit Label/Value Pairs dialog box appears. Use the dialog box to specify the label and value associated with each cross-bar position on the slide control. A slide control can have up to 32 labels and associated values.

Create»Binary

A binary control operates like a mechanical on/off switch. A binary control gives a parameter value one of two predefined values, depending on whether the control is in the on or off position. When you select **Create»Binary**, the Create Binary Control dialog box appears.

The following items appear in the Create Binary Control dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Parameter Position**—The location of the control value in the function parameter list. For a control in a common control function panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a Function Panel window. The first position is 1.

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control function panel. The first position after the controls in the common control function panel is 1. If there is no common control function panel, the first position is one 1.

- **Data Type**—The data type of the values in the binary control.

- **Default Value**—The default for the binary control. The default can be **On** or **Off**.

- When you select the **On/Off Settings** button, the Edit On/Off Settings dialog box appears.

  – **ON Text**—The label that appears next to the upper (on) position of the binary control.

  – **OFF Text**—The label that appears next to the lower (off) position of the binary control.

  – **ON Value**—The value, constant name, or valid C expression you want to associate with the **On** label.

  – **OFF Value**—The value, constant name, or valid C expression you want to associate with the **Off** label.

Create»Ring

A ring control shows the user an option list. A ring control displays only one item at a time from its list of options. When you select **Create»Ring**, the Create Ring Control dialog box appears.

The following items appear in the Create Ring Control dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Parameter Position**—The location of the control value in the function parameter list. For a control in a common control function panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a Function Panel window. The first position is 1.

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control function panel. The first position after the controls in the common control function panel is 1. If there is no common control function panel, the first position is 1.

- **Data Type**—The data type of the values in the ring control.

- **Default Value**—The default for the ring control. The default must be one of the labels specified in the Edit Label/Value Pairs dialog box.

- **Control Width**—The width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

- When you click the **Edit Label/Value Pairs** button, the Edit Label/Value Pairs dialog box appears. Use dialog box to specify the label and value associated with each entry in the ring control. A ring control can have up to 32,000 labels and associated values.

Create»Numeric

A numeric control is an input control that lets you increment or decrement a numeric value using the up and down arrows. When you select **Create»Numeric**, the Create Numeric Control dialog box appears.

The following items appear in the Create Numeric Control dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Parameter Position**—The location of the control value in the function parameter list. For a control in a common control function panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a Function Panel window. The first position is 1.

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control function panel. The first position after the controls in the common control function panel is 1. If there is no common function control panel, the first position is 1.

- **Data Type**—The data type of the values in the numeric control. You can choose from the following data types:

  ```
  int
  short
  char
  unsigned int
  unsigned short
  unsigned char
  double
  float
  ```

  You also can choose a user-defined data type for which you have specified an intrinsic type.

- **Default Value**—The default for the numeric control, which must be a valid member of the value set.

- **Display Format**—The output format. You can display integers, longs, and shorts in decimal, hexadecimal, octal, or ASCII format. You can display doubles and floats in either scientific or floating-point notation.

- **Precision**—The number of digits the control displays to the right of the decimal point.

- When you click the **Value Set** button, the Edit Value Set dialog box appears.

  – **Minimum**—The minimum value the numeric control accepts.

  – **Maximum**—The maximum value the numeric control accepts.

  – **Inc Value**—The amount the numeric control value increments or decrements when the user presses the up or down arrows. The value in **Inc Value** must divide evenly into the range of the numeric control.

Create»Output

An output control displays the results of a function call. When you select **Create»Output**, the Create Output Control dialog box appears.

The following items appear in the Create Output Control dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Parameter Position**—The location of the control value in the function parameter list. For a control in a common control function panel, **Parameter Position** specifies the control value in the parameter lists of all function panels in a Function Panel window. The first position is 1.

  For a control on a function panel, **Parameter Position** specifies the control value in the parameter list after the controls in the common control function panel. The first position after the controls in the common control function panel is 1. If there is no common control function panel, the first position is 1.

- **Data Type**—The data type of the variable or value displayed in the output control.

- **Default Value**—The default value that the control displays. You can leave this item blank.

- **Display Format**—The format in which the output control displays values. You can display integers, longs, shorts, and chars in decimal, hexadecimal, octal, or ASCII format. You can display doubles and floats in either scientific or floating-point notation. If the data type is `char *`, `void *`, a meta data type, or an array, the display format control is not valid.

- **Control Width**—The width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Return Value

A return value control displays a value returned from a function. You can use a return value control only if the function has a non-void data type. When you select **Create»Return Value**, the Create Return Value Control dialog box appears.

The following items appear in the Create Return Value Control dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Data Type**—The data type of the variable or value displayed in the return value control. The data type can be any data type other than an array type or a meta data type.

- **Display Format**—The format in which the return value control displays values. You can display integers, longs, shorts, and chars in decimal, hexadecimal, octal, or ASCII format. You can display doubles and floats in either scientific or floating-point notation. If the data type is `char *` or `void *`, the display format control is not valid.

- **Control Width**—The width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Global Variable

A global variable control displays the value of a global variable defined in LabWindows/CVI when users operate the function panel. When you select **Create»Global Variable**, the Create Global Variable Control dialog box appears.

The following items appear in the Create Global Variable Control dialog box.

- **Control Label**—The label that appears above the control on the function panel.

- **Global Variable Name**—The name of the variable whose contents are shown in the global control.

- **Data Type**—The data type of the item entered in the global variable control.

- **Display Format**—The format in which the global variable control displays values. You can display integers, longs, shorts, and chars in decimal, hexadecimal, octal, or ASCII format. You can display doubles and floats in either scientific or floating-point notation. If the data type is `char *`, `void *`, a meta data type, or an array, the display format control is not valid.

- **Control Width**—The width of the control in pixels. The minimum allowed is 24. The maximum allowed is 2,048. The default control width is 145 pixels.

Create»Message

You can place text anywhere on the panel with a message control. The message control serves as an online documentation tool for panels. When you select **Create»Message**, a dialog box appears. Enter the text into the message text control and click **OK**. To enter a new line in the message text control, press <Ctrl-Enter>. The text appears on the panel, and you can position it like any other control.

Create»Function Panel

Use this command to create multiple function panels within one Function Panel window.

Create»Common Control Function Panel

Use this command to create a common control function panel. The *n* controls on a common control function panel specify the first *n* parameters of all functions in the Function Panel window.

View Menu for the Function Panel Editor

The **View** menu for the Function Panel Editor works in the same way as the **View** menu for the Function Panel windows. For more information about these commands, refer to the *View Menu for Function Panel Windows* section of Chapter 6, *Using Function Panels*.

View»Panels

Use this command to cycle through the multiple function panels in the Function Panel window.

Instrument Menu for the Function Panel Editor

The **Instrument** menu for the Function Panel Editor works in the same way as the **Instrument** menu for the Workspace window. For more information about these commands, refer to the *Instrument Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Tools Menu for the Function Panel Editor

Many of the commands in the **Tools** menu for the Function Panel Editor work in the same way as the **Tools** menu for the Function Tree Editor. For more information about these commands, refer to the *Tools Menu for the Function Tree Editor* section of Chapter 7, *Function Tree Editor*.

Tools»Generate Source for Function Panel

Generate Source for Function Panel generates function definitions and declarations in your driver source and header files. If a function definition already exists, LabWindows/CVI prompts you for permission to update it. You can replace, insert above or below, or skip without updating. Your choice also applies to the declaration.

Window Menu for the Function Panel Editor

The **Window** menu for the Function Panel Editor works in the same way as the **Window** menu for the Workspace window. For more information about these commands, refer to the *Window Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Options Menu for the Function Panel Editor

Use the commands in the **Options** menu to invoke the Function Tree Editor, operate the function panel, or toggle the scroll bars.

Options»Data Types

Use the **Data Types** command to specify the names of user-defined data types. Data types you specify with the **Data Types** command appear in the Data Type ring control on the Edit Control dialog boxes for input, slide, binary, ring, output, and global variable controls.

📝 **Note**   The .h file for the instrument driver must define the types that you specify with the **Data Types** command.

When you select **Options»Data Types,** the Edit Data Type List dialog box appears. The items in the Edit Data Type List dialog box are as follows:

- **Type**—Specifies the name of a user-defined data type.
- The list box in the dialog box displays the currently available user-defined data types.
- **Intrinsic Data Type**—Allows you to associate each user-defined data type with one of the intrinsic C data types that you can use in a numeric control. If you select an item other than **None**, you can use the user-defined data type as the data type for a numeric control.
- **Add**—Places the name in the **Type** control in the data type list.
- **Move Up**—Moves the selected entry up one line in the data type list.
- **Move Down**—Moves the selected entry down one line in the data type list.
- **Change**—Displays a dialog box that prompts you to change the selected entry in the data type list.
- **Delete**—Removes an entry in the data type list.
- **Add VISA Types**—Adds the special set of data types defined by the NI-VISA Library.
- **Done**—Accepts edits to the data type list and returns to the Function Panel Editor.

Options»Toolbar

Use this command to open the Customize Toolbars dialog box. For more information about customizing the toolbar, refer to the *Toolbars in LabWindows/CVI* section of Chapter 2, *Workspace Window*.

Options»Initial Control Width

With the **Initial Control Width** command, you can set the initial width for all input, output, ring, return value, and global variable controls that you create in the Function Panel window. The default initial control width is 145 pixels. LabWindows/CVI saves your settings between sessions.

Options»Grid Line Options

Use this command to specify whether grid lines appear in the Function Panel Editor and whether you want the controls to snap to the grid lines. To customize the appearance and spacing of the grid lines, select **Options»Preferences** in the User Interface Editor.

Options»Revert to Default Panel Size

The **Revert to Default Panel Size** command sizes and positions the function panel so that it fills up the exact default Function Panel window size.

Options»Panels Movable

Use the **Panels Movable** command to specify whether panels are movable within the Function Panel Editor. Panels are never movable in operate mode.

Options»Toggle Scroll Bars

The **Toggle Scroll Bars** command adds or removes horizontal and vertical scroll bars from a function panel.

Options»Open Function Panels in New Window

Use this command to open a new Function Panel window for every function panel you open.

Options»Edit Function Tree

The **Edit Function Tree** command opens the Function Tree Editor.

Options»Operate Function Panel

Use the **Operate Function Panel** command to operate the current Function Panel window.

Options»Save in XML Format

This command works in the same way as the command for the Function Tree Editor. Refer to the *Options Menu for the Function Tree Editor* section of Chapter 7, *Function Tree Editor*.

Options»Load from XML Format

This command works in the same way as the command for the Function Tree Editor. Refer to the *Options Menu for the Function Tree Editor* section of Chapter 7, *Function Tree Editor*.

Help Menu for the Function Panel Editor

The **Help** menu for the Function Panel Editor works in the same way as the **Help** menu for the Workspace window. For more information about these commands, refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Creating a Function Panel Window

In this example, you create a function panel without writing any code. The example panel controls an oscilloscope with two channels and configures the vertical sensitivity, coupling, and invert setting of the oscilloscope.

Complete the following steps to create a new instrument and panel.

1. Select **File»New»Function Tree (*.fp)**.
2. Select **Create»Instrument**.
3. Enter `Function Panel Examples` as the **Name** and `panel` as the **Prefix**. Click **OK**.
4. Select **Create»Function Panel Window**.
5. Enter `Configure` as the **Name** and `config` as the **Function Name**. Click **OK**.
6. Double-click the `Configure` node in the function tree. A new Function Panel window that contains a single function panel appears on the screen. Notice that the code name of the function appears in the generated code window, preceded by the prefix.

7. Select **Create»Binary** and complete the Create Binary Control dialog box, except the **Default Value**, as shown in the following figure.
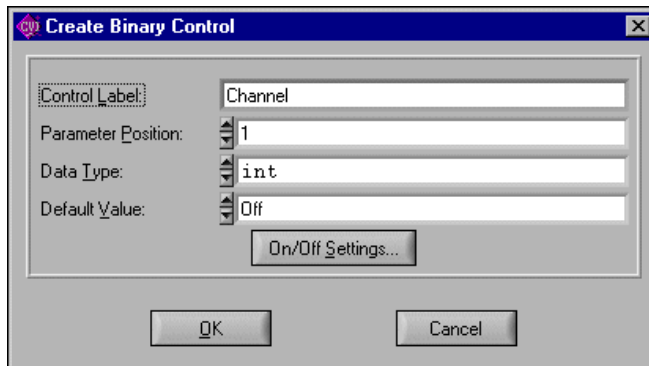
8. Click the **On/Off Settings** button and complete the Edit On/Off Settings dialog box as shown in the following figure. Click **OK** in both dialog boxes and position the control on the panel.

9. In the Create Binary Control dialog box, set the **Default Value** to 1.
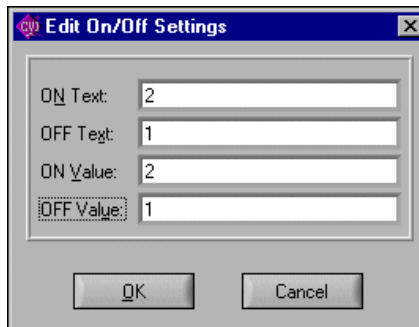
10. Select **Create»Input** and complete the Create Input Control dialog box as shown in the following figure. Click **OK** and position the control on the panel.
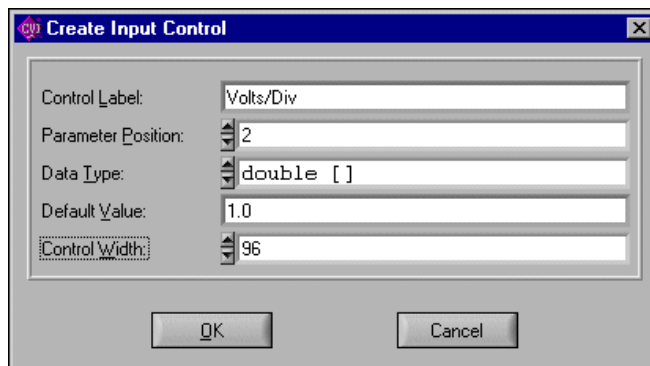
11. Select **Create»Slide**.

12. Complete the Create Slide Control dialog box as shown in the following figure.
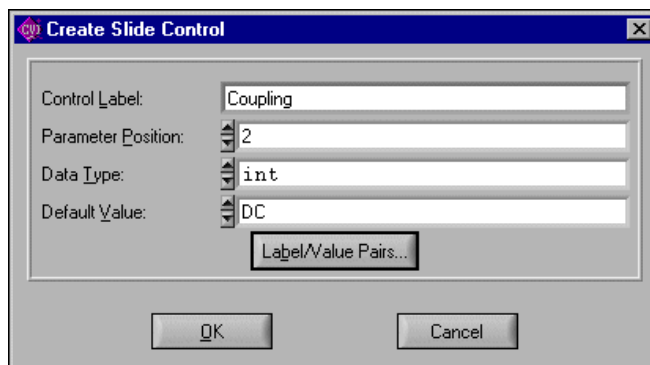
13. Click **Label/Value Pairs** and complete the Edit Label/Value Pairs dialog box as shown in the following figure. Click **OK** in both dialog boxes and position the control on the panel.
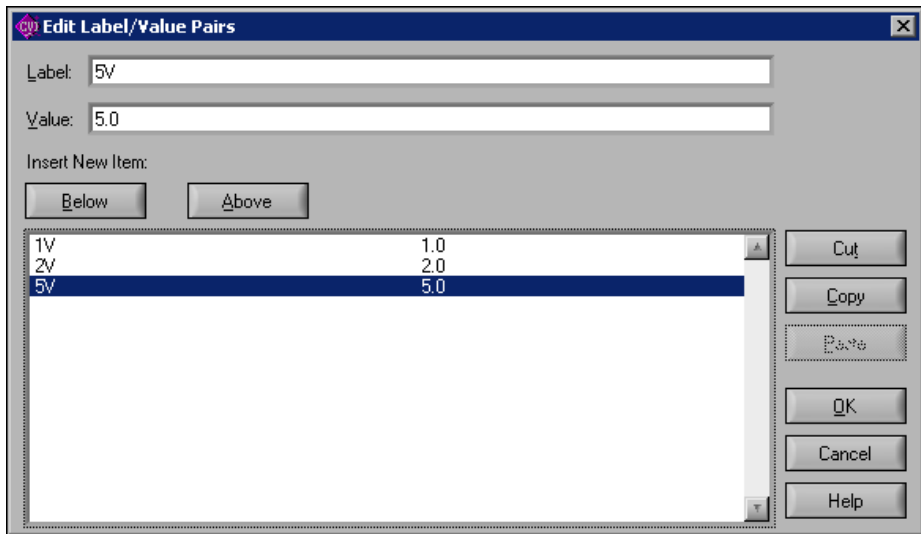
14. Select **Create»Binary** and complete the Create Binary Control dialog box as shown in the following figure.
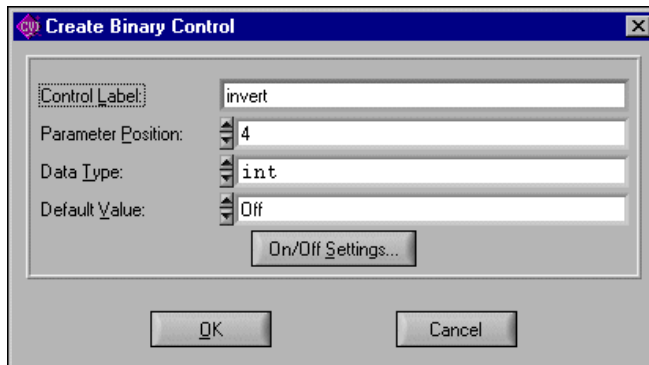
15. Click the **On/Off Settings** button and complete the Edit On/Off Settings dialog box as shown in the following figure. Click **OK** in both dialog boxes and position the control on the panel.
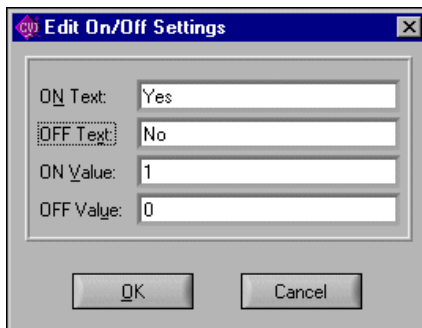
Changing Control Type

In this example, you change the type of the **Volts/Div** control from an input control to a slide control. Complete the following steps:

1. Make sure the Function Panel window from the previous example is active and in edit mode. Place your cursor on the **Volts/Div** control.

2. Select **Edit»Change Control Type**.

3. In the Change Control Type dialog box, select **Slide** and click **OK**. The Edit Slide Control dialog box appears.

4. Click **Label/Value Pairs**. The Edit Label/Value Pairs dialog box appears.

5.   Complete the dialog box as shown in the following figure and click **OK**.
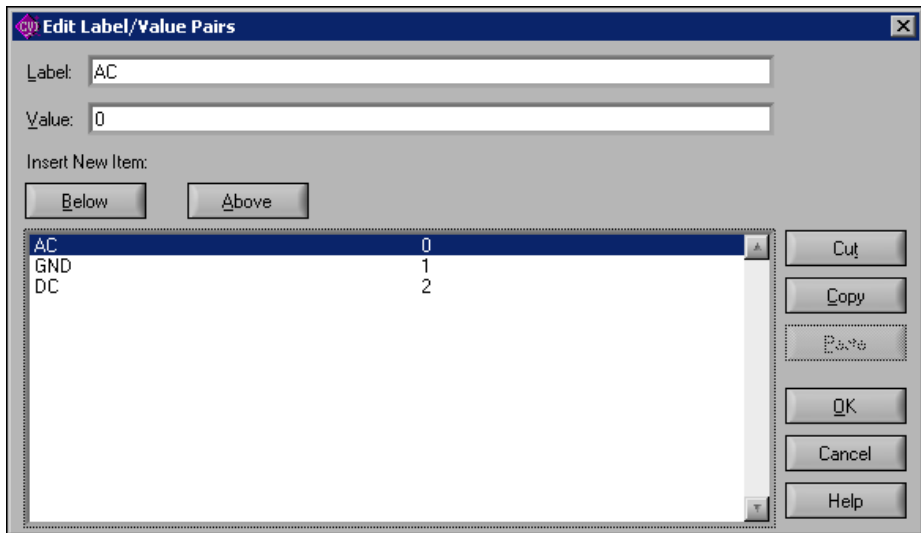
6.   Click **OK** in the Edit Slide Control dialog box to replace the **Volts/Div** input control with a slide control.

Suppose that you meant this control to be a ring control instead of a slide control. Complete the following steps:

1.   Place your cursor on the **Volts/Div** control.

2.   Select **Edit»Change Control Type**.

3.   Select **Ring**. The Edit Ring Control dialog box appears.

4.   Click **Label/Value Pairs**, leaving all other items unchanged. The Edit Label/Value Pairs dialog box appears. Notice that the slide control label value pairs remain. Click **OK**.

5.   Click **OK** again to close the Edit Ring Control dialog box.

A ring control replaces the **Volts/Div** slide control on the function panel.

Cutting and Pasting Controls

You frequently might want to cut and paste controls. In this example, you copy controls from one panel to another. Complete the following steps to copy a control:

1. Make sure the function panel from the previous example is active and in edit mode. Position the cursor on the **Volts/Div** control and select **Edit»Control Help**.

    Enter the following text in the Help Editor:

    ```
    This control specifies the volts per division setting of the
    oscilloscope.
    ```

2. Save the .fp as example.fp and select **File»Close** in the Help Editor dialog box.

3. With the selection still on the **Volts/Div** control, select **Edit»Copy Controls**.

4. Select **Edit»Paste**.

5. With the selection on the new control, select **Edit»Edit Control**.

6. Change the ring **Control Label** to Volts/Div 2 and the **Parameter Position** to 2.

Notice in the Generated Code window that the config function now has an additional parameter, **Volts/Div 2**.

Complete the following steps to create a new function panel and copy a control to the panel.

1. Select **Options»Edit Function Tree**.

2. Create a Function Panel window with the following parameters. Type New Panel in the **Name** box and new_panel in the **Function Name** box.

3. Highlight the Configure node in the Function Tree Editor.

4. Select **Edit»Edit Function Panel Window** to return to the Configure panel.

5. Position the selection on the control **Volts/Div 2**.

6. Select **Edit»Cut Controls**.

7. Press <Ctrl-Page Down> to move to the New Panel function panel.

8. Select **Edit»Paste**.

The control appears on the panel. View the help by selecting **Edit»Control Help**. Notice that the help is copied with the control.

9

Adding Help to Instrument Drivers

Use the Help Editor to add help to instrument drivers that you create. The Help Editor is a text editor in which you can enter help, with HTML tags or without. The Help Editor also provides menu options to cut and copy text, undo the last action, search within the text, and so on. This section describes the types of help available from an instrument driver and how you can create help.

New Style versus Old Style Help

LabWindows/CVI has two styles of online help for instrument drivers: new (recommended) and old (LabWindows DOS). The old help style maintains compatibility with help created in LabWindows version 2.3 or earlier. This help style uses the DOS/IBM character set so it can display special extended ASCII characters used by older instrument drivers.

The new help style uses the standard Windows character set and automatically displays the control help with control name and data type information. You also can include HTML tags in the help.

There is also a difference in the type of help that can be displayed. In either new or old style help, you can view instrument help, function class help, and control help. However, the help for functions appears differently between the two styles. This difference is noticeable only when you have multiple function panels on a single Function Panel window. In the new style, you can access function help for each function panel. In the old style, you can access the Function Panel window help, which describes all the functions contained in that Function Panel window.

National Instruments recommends that you use the new help style for all help for instrument drivers that you create in LabWindows/CVI. Most of the discussion in this chapter assumes you are using the new style help.

Help Options

The user of an instrument driver can view the help listed in the following table.

Type of Help	Location of Help
Instrument help	Access instrument help by right-clicking an instrument in the Instruments folder and selecting **Instrument Help**.
Function class help	Access class help by right-clicking a class and selecting **Class Help**.
Function help (New style help only)	Access function help by selecting **Help»Function** in the function panel.
Function Panel window help	Access help through a dialog box that appears when you select an instrument from the **Instrument** menu. This help is directly editable only in old style help. In the new style help, it is generated from the function help for each function in the window.
Control help	Select **Help»Control** in the function panel to view help for the individual controls on the function panel.

Editing Help

There are four types of help information that you can enter: instrument, class, function, and control. You can edit instrument and class help from the Function Tree Editor and edit function and control help from the Function Panel Editor. Each of the editors has an **Edit** menu. Select **Edit»Edit Help** in the Function Tree Editor to add instrument and class help. Select **Edit»Function Help** and **Edit»Control Help** in the Function Panel Editor to add function panel and control help.

Complete the following steps to add help.

1. From either the Function Tree Editor or the Function Panel Editor, select the instrument driver, class, function, or control that you want to add help to.

2. Select **Edit»Edit Help**, **Edit»Node**, **Edit»Function Help**, or **Edit»Control Help**. The Help Editor window appears.

    You also can right-click the instrument driver or class in the Function Tree Editor and select **Edit Help** from the context menu. You can right-click the panel in the Function

Panel Editor and select **Function Help**. In the Function Panel Editor, right-click the control and select **Control Help** to edit help for individual controls.

3. Enter help in the text box.

Viewing Instrument Driver Help

You can access help for instrument drivers in the following ways:

• To view instrument driver help, select **Instrument Help** in the Class Help panel, accessible through the Select Function Panel dialog box. You also can right-click the instrument driver in the Library Tree and select **Instrument Help** from the context menu.

• To view class help, select the class in the Select Function Panel dialog box and click the **Help** button. You also can right-click the class in the Library Tree and select **Class Help** from the context menu.

• To view new style function help, select **Help»Function** in the function panel or right-click the function panel background. You also can right-click the function panel or right-click the function in the Library Tree and select **Function Help** from the context menu. Viewing old style help differs.

• To view control help, select **Help»Control** in the function panel. You also can right-click the control.

Function Class Help

To display help about a class of function panels, right-click the class in the function tree and select **Class Help** or select the class in the Select Function Panel dialog box and click the **Help** button.

Enter function class help from the Function Tree Editor. Complete the following steps to add help information.

1. Select the class node in the function tree.

2. Select **Edit»Edit Help** in the Function Tree Editor. The Help Editor appears. Alternatively, you can right-click the class node and select **Edit Help** to display the Help Editor.

3. Enter the help text into the Help Editor.

Control Help

To display help for a specific function panel control, select the control and select **Help»Control** or right-click the control.

You can enter control help from the Function Panel Editor.

Complete the following steps to add help for a function panel control.

1. Select the control.

2. Select **Edit»Control Help**. The Help Editor appears. Alternatively, you can right-click the control to display the Help Editor.

3. Enter the help text into the Help Editor. You can add HTML tags to the text, or you can leave out the HTML tags.

Function Panel Window Help (Old Style Help Only)

When you use the old help style, you can display help that pertains to a Function Panel window by selecting **Help»Window** in the function panel. Alternatively, you can right-click the background the Function Panel window to display the function panel help.

When you use the old help style, enter Function Panel window help information from the Function Panel Editor. Complete the following steps to add Function Panel window help.

1. Select **Edit»Window Help** in the Function Panel Editor. The Help Editor appears. Alternatively, you can right-click the background of the Function Panel window to display the Help Editor.

2. Enter the help text into the Help Editor.

File Menu for the Help Editor

The commands in the **File** menu for the Help Editor work in the same way as the commands in the **File** menu for the Workspace window. Refer to the *File Menu for the User Interface Editor* section of Chapter 3, *User Interface Editor*, for more information.

File»Add Program File to Project

The **Add Program File to Project** command adds the instrument driver program file associated with the `.fp` file of the current Function Panel window to the project list.

Edit Menu for the Help Editor

Use the commands in the **Edit** menu to edit the help text in the window. Many of these commands work in the same way as the commands in the **Edit** menu for the Source window. Refer to the *Edit Menu for the Source and Interactive Execution Windows* section of Chapter 4, *Source and Interactive Execution Windows*, for more information.

Edit»Paste

Paste inserts the contents of the clipboard into the window at the location of the cursor.

Edit»Delete

Delete discards the selected text in the window without copying it to the clipboard.

Edit»Find

Find locates a particular text string in the Help Editor.

Edit»Replace

Replace replaces particular text in the Help Editor with text you specify.

Edit»Revert

Revert returns the most recently saved version of help text to the window.

Tools Menu for the Help Editor

Use the commands in the **Tools** menu to jump back to the function panel or function tree node that the help text in the window applies to. Refer to the *Tools Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for information about the **Customize** command.

Tools»Edit Function Tree

Edit Function Tree opens the Function Tree Editor and highlights the function tree node that contains the current help text.

Tools»Edit Function Panel

Edit Function Panel opens the Function Panel Editor for the function panel that contains the current help text. If the help text applies to a particular control on the function panel, the **Edit Function Panel** command selects the control.

Window Menu for the Help Editor

The **Window** menu for the Help Editor works in the same way as the **Window** menu for the Workspace Window. For more information about these commands, refer to the *Window Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

Help Menu for the Help Editor

Use the commands in the **Help** menu to access information about LabWindows/CVI. These commands work in the same way as the **Help** menu commands for the Workspace window. Refer to the refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Adding Help in the Function Tree Editor

In this example, you will add instrument and function class help information to a function tree. Complete the following steps to create a new instrument and function tree and add help information to the function tree.

1. Select **File»New»Function Tree (*.fp)**.

2. Select **Create»Instrument**.

3. Enter `Help Examples` as the **Name** and `help` as the **Prefix**. Click **OK**.

4. Select **Create»Class**.

5. Enter `Class 1` as the **Name**. Click **OK**.

6. Select the line beneath `Class 1`.

7. Select **Create»Function Panel Window**.

8. Enter `Function 1` as the **Name** and `fun1` as the **Function Name**. Click **OK**.

9. The first level of help is associated with the name of the instrument driver. Right-click **Help Examples** and select **Edit Help** to open the Help Editor.

10. Enter the following texts:

    ```
    This driver was created to illustrate how to add help text to an
    instrument driver.
    ```

11. Select **File»Save .FP File**. Save the file as `help.fp`. Close the Help Editor.

12. Right-click `Class 1` and select **Edit Help**.

13. Enter the following text in the Help Editor.

    ```
    An example function class. The functions in this class are the
    following:
    Function 1--The only function in the class.
    ```

14. Save the `.fp` file and close the Help Editor.

Complete the following steps to view the help.

1. Select **Instrument»Help Examples**. The Select Function Panel dialog box appears.

2. Select `Class 1` and click **Help** to open the Class Help window.

3. Click **Instrument Help** to display the Instrument Help window.

4. Click **Done** to exit the Instrument Help and Class Help windows.

5. Click **Cancel** to exit the Select Function Panel dialog box.

Adding Help in the Function Panel Editor

In this example, you will add help to function panels and function panel controls from the Function Panel Editor. Double-click `Function 1` from the previous example.

Complete the following steps in the Function Panel Editor to modify the help for the function panel.

1. Select **Edit»Function Help** from the **Edit** menu. The Help Editor window appears.

2. Enter the following text:

   `This function is the only function in Class.`

3. Select **File»Save .FP File** then **File»Close** to save the text and remove the Help Editor.

Help also is associated with each of the controls in a function.

Complete the following steps to add a control to the current panel.

1. Select **Create»Input**.

2. Enter `Input Control` for the **Control Label**.

3. Click **OK**.

Complete the following steps to add help to the control.

1. Select the control and select **Edit»Control Help**. Alternatively, right-click the control and select **Control Help**. The Help Editor appears.

2. Enter the following text in the Help Editor.

   `This control is an input control on the Function 1 function panel.`

3. Select **File»Save .FP File** then **File»Close** to save the text and remove the Help Editor.

You have now added help to all possible locations. Select **Options»Operate Function Panel** and then view the help for the function panel.

Copying and Pasting Help Text

In this exercise, you will copy text between function panels, controls, and instruments. The clipboard retains its contents as you move between controls, function panels, and even instruments. Help text also stays with a control or function panel that is cut, copied, or pasted.

Complete the following steps to copy the help between controls on different panels.

1. Create a new Function Panel window for `help.fp`. Enter `Function 2` in the **Name** box and `fun2` in the **Function Name** box.

2. The `Function 1` function panel from the previous example should be on the screen in **Edit** mode. Double-click `Function 1` in the Function Tree Editor.

3. Select **Create»Global Variable**.

4. Enter `Status` in **Control Label** and `ibsta` in the **Global Variable Name**. Leave all other items at their default settings. Click **OK**.

5. Add the following help to the global control.

   ```
   This control displays the status of GPIB function calls.


   Errors:

   0              Success

   non-zero       See the Status control on any GPIB
                  Library function panel
   ```

6. Select **File»Save .FP file** and then **File»Close** to save the text and exit the Help Editor.

7. Select the **Status** control. Select **Edit»Copy Controls**.

8. Press <Ctrl-Page Down> to display the `Function 2` function panel.

9. Select **Edit»Paste**. The **Status** control appears on the function panel.

10. Select **Options»Operate Function Panel** and view the help. Notice that the help stays with a control when you copy that control.

Complete the following steps to copy the help text without copying the control.

1. Select **Options»Edit Function Panel**.

2. Select **Create»Global Variable**.

3. Complete the Create Global Variable Control dialog box as follows. Type `Error` in **Control Label** and `iberr` in **Global Variable Name**. Leave all other items at their default settings. Click **OK**.

4. Select the **Status** control.

5. Select **Edit»Control Help** or right-click the control and select **Control Help**.

6. Select all the text in the dialog box.

7. Select **Edit»Copy**.

8. Select **File»Close**.

9. Select the **Error** control.

10. Select **Edit»Control Help** or right-click the control and select **Control Help**.

11. Select **Edit»Paste**. The help appears in the window.

12. Modify the text so it reads as follows:

```
This control displays the value of the GPIB global error variable.

The control displays the value of the error only when the Status
control is non-zero.


Errors:

0               Success

non-zero        See the Status control on any GPIB Library
                function panel
```

In these examples, you have learned to copy or move text from one control to another. Use the same methods to copy and move help text between various locations. For example, use these methods to copy and move panel, instrument, window, and control help within an instrument driver or across instrument drivers.

10

Variables and Watch Windows

Variables Windows

Use the Variables window to inspect and modify the values of program variables. You can open this window when a program is suspended at a breakpoint or at any other time.

The Variables window shows the names and types of all variables, including arrays and strings. The current values of numeric scalars, values and contents of pointers, and string contents appear in the Variables window.

✎ **Note**   When strings appear in ASCII format, there is no visual distinction between a space (ASCII 32) and a NUL byte (ASCII 0). To see the difference, select **Format»Decimal** to view the string in decimal format.

To view the Variables window, select **Window»Variables** in the active LabWindows/CVI window. While the program is suspended, you can open the Variables window for the currently highlighted variable from a Source or function panel by selecting **Run»View Variable Value** in the Source window or **Code»View Variable Value** in the function panel.

The Variables window shows all currently defined variables in LabWindows/CVI. LabWindows/CVI updates variables in this window at each breakpoint. The vertical bars separate the window into three scrollable fields: name, value, and variable type. You can change the width of the fields by dragging the vertical bars with the mouse. The window also is divided into two horizontal sections: the Global subwindow and the function subwindow.

The Global subwindow displays the following variables:

*   Project globals, including all global variables not declared as `static`
*   Interactive Execution window variables declared in the Interactive Execution window
*   Global variables declared as `static`

The function subwindow displays function parameters and local variables from currently active functions. The variable list for each function is grouped in a different section. For any given function, the Variables window lists formal parameters first followed by local variables. Formal parameters appear in italics.

The following icons appear to the left of certain variables.

The variable on this line is the starting pointer to a block of defined data such as an array, string, or structure. Click this icon or select **View»Expand Variable** to expand the variable so that you can see each element or member.

The variable on this line is the starting pointer to a block of defined data that appears in expanded form. Click this icon or select **View»Close Variable** to close the variable so that you see only the starting pointer.

The variable on this line is a member of a structure that is a parent pointer to another structure of the same type. Click this icon or select **View»Follow Pointer Chain** to replace the current structure with the child structure that the pointer references.

The variable on this line is a child structure in a chain. The pointer to its parent structure does not appear. Click this icon or select **View»Retrace Pointer Chain** to replace the current structure with its parent.

You can view Variable window variables in the Graphical Array View, String Display, Array Display, and Memory Display windows. Select the text you want to view from the Variables window and drag it into the windows.

Watch Window

The Watch window is similar in nature to the Variables window except that you can select your own set of variables and expressions to view in the Watch window. By default, LabWindows/CVI updates variables and expressions in the Watch window at each breakpoint, but you also can set them to update continuously and cause a breakpoint when their values change. To open the Watch window, select **Window»Watch** in the active LabWindows/CVI window.

Select Watch window variables from the Variables window using the **Options»Add Watch Expression** command. The **Edit»Add Watch Expression** command opens the Add/Edit Watch Expression dialog box.

You also can drop text from the Source and Interactive Execution windows into the Watch window to add a watch expression. When you drop text into the Watch window, the Add/Edit Watch Expression dialog box appears. If you drop code into the lower half of the Watch window, LabWindows/CVI sets the **Scope** of the variable or expression as **Local**. If you drop code into the upper half of the Watch window, LabWindows/CVI sets the **Scope** of the variable or expression as **Global to project/DLL**.

You can view Watch window variables and expressions in the Graphical Array View, String Display, Array Display, and Memory Display windows. Select the text you want to view and drag it into the windows.

File Menu for Variables and Watch Windows

The commands in the **File** menu work in the same way as the **File** menu commands for the Workspace window. Refer to the *File Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

File»Output

The **Output** command writes the contents of the window to an ASCII file on disk. When you select **Output**, a dialog box appears prompting you to specify the name of the file.

File»Hide

The **Hide** command visually closes a window while retaining the contents in memory.

Edit Menu for Variables and Watch Windows

This section contains a detailed description of the **Edit** menu for the Variables and Watch windows.

Edit»Edit Value

Use the **Edit Value** command to change the value of a variable. When you select **Edit»Edit Value** or double-click the variable name, the Edit Value dialog box appears in which you can type the new value.

The value that you enter in the Edit Value dialog box depends on the type and display format of the variable, as the following instructions demonstrate:

- Edit integers and longs in the format in which they appear.
- Edit real numbers in either scientific or floating-point format, regardless of the display format.
- Edit individual array elements by expanding the array using the **View»Expand Variable** command.
- Edit individual bytes of a string by expanding the string using the **View»Expand Variable** command. The bytes appear in the integer format you specify in the **Format** menu.

Edit»Find

The **Find** command invokes the Find dialog box.

Enter the text you want to find in **Find What**. If you select text on a single line before you execute the **Find** command, the selected text appears in **Find What**. Otherwise, the text you last searched for appears in the box. You can access a history of selections for **Find What** by clicking the arrow to the right of **Find What** or by using the up or down arrow keys on your keyboard. The Find dialog box contains the following options:

- **Case Sensitive**—Finds only the instances of the specified text that match exactly. For example, if CHR is the specified text, the **Case Sensitive** option finds CHR but not Chr.

- **Whole Word**—Finds the specified text only when the characters that surround it are spaces, punctuation marks, or other characters not considered parts of a word. LabWindows/CVI treats the characters A through Z, a through z, 0 through 9, and underscore (_) as parts of a word.

- **Wrap**—Specifies to continue searching from the beginning of the window once the end of the window has been reached.

- **Regular Expression**—If you select this option, LabWindows/CVI treats certain characters in **Find What** as regular expression characters instead of literal characters. Refer to the *Regular Expression Characters* section in Chapter 5, *Source and Interactive Execution Windows*, for a list of regular expression characters.

- **Name**—Includes the variable name field of the Variables/Watch window in the search.

- **Value**—Includes the value field of the Variables/Watch window in the search.

- **Type**—Includes the variable type field of the Variables/Watch window in the search.

- **Button Bar**—Enables or disables the built-in dialog box for interactive searching.

  Find Prev and **Find Next** search for the closest previous or next occurrence of the specified text. **Stop** terminates the search, leaving the highlight on the current line. **Return** terminates the search, moving the highlight to where you initiated the search.

The search shortcut keys remain active even if you disable the Button Bar. The search shortcut keys in the Variables and Watch windows are the same as the search shortcut keys in Source windows. Use the **Options»Change Shortcut Keys** command in the Source window for a list of the default search shortcut keys.

Edit»Next Scope

In the function subwindow, **Next Scope** highlights the function that called the current function. In the Global subwindow, **Next Scope** highlights the next module.

This command appears only in the Variables window.

Edit»Previous Scope

In the function subwindow, **Previous Scope** highlights the function that the current function called directly. In the Global subwindow, **Previous Scope** highlights the previous module.

This command appears only in the Variables window.

Edit»Add/Edit Watch Expression

These commands do not appear only in the Variables window. The Add/Edit Watch Expression dialog box has the following options:

- **Variable/Expression**—Contains the variable or expression to place in the Watch window.

- **Scope**—Corresponds to whether the variable or expression variables are global to the project, global to a file, local to a function, or global to the Interactive Execution window.

- **Executable/DLL**—Indicates the executable or DLL to which the watch expression applies. The default value for the control is the debug executable or DLL name for the active project. When you start debugging a project, LabWindows/CVI changes an empty string to the name of the debug executable or DLL for the current project. The menu ring to the right of the control contains the names of all debuggable executables and DLLs in the workspace. If you want the watch expression to apply to a DLL that is not in the workspace, you must supply the name of the DLL. Enter the filename and extension without a directory path, such as `mydll.dll`. To set a watch expression for a DLL, it is easiest to first set a breakpoint in a DLL source file. Once the DLL has been loaded and program execution suspends, select the DLL name from the menu ring.

- **File**—Contains the name of the file that defines the variable or expression variables if they are global to a file or local to a function.

- **Function**—Contains the name of the function that defines the variable or expression variables if they are local to a function.

- **Update display continuously**—Causes the variable or expression to be evaluated and updated in the Watch window between each statement in your program while the program is running.

- **Break when value changes**—Suspends the program when the value of the variable or expression changes.

- **Replace**—Replaces the previous attributes of the current variable or expression of the same name in the Watch window with the current attributes of the dialog box. **Replace** is available only when you invoke the dialog box from the Watch window.

- **Add**—Inserts the variable or expression into the Watch window.

- **Cancel**—Aborts the operation.

You can add watch expressions to the Watch window directly from a Source window or a function panel. To add a watch expression from a Source window, highlight the expression and select **Run»Add Watch Expression**. To add a watch expression from a function panel, highlight the expression and select **Code»Add Watch Expression**.

You also can drop text from the Source, Interactive Execution, or Variables window into the Watch window to add a watch expression. When you drop text into the Watch window, the Add/Edit Watch Expression dialog box appears. If you drop code onto the lower half of the Watch window, LabWindows/CVI sets the **Scope** of the variable or expression as **Local**. If you drop code onto the upper half of the Watch window, LabWindows/CVI sets the **Scope** of the variable or expression as **Global to project/DLL**.

Edit»Delete Watch Expression

Delete Watch Expression removes the selected watch variable/expression from the Watch window. This command is not available in the Variables window.

View Menu for the Variables and Watch Windows

This section contains a detailed description of the **View** menu for the Variables and Watch windows.

To use one of these commands, select a particular variable or expression by clicking it or using the up and down arrow keys and then accessing the command from the **View** menu.

View»Expand Variable

The Variables and Watch windows can display arrays, strings, and structures in closed form or expanded form. In closed form, you see only the name and address of the aggregate variable next to the triangle icon.

In expanded form, the icon changes to a circle, and you see the individual elements and their values.

The **Expand Variable** command expands a currently closed aggregate variable so you can see its contents. Clicking the triangle icon has the same effect as selecting **View»Expand Variable**.

View»Close Variable

The **View»Close Variable** command closes the currently expanded aggregate variable so you can see its name and starting address. Clicking the circle icon has the same effect as selecting **View»Close Variable**.

View»Follow Pointer Chain

Use **Follow Pointer Chain** to examine complex pointer-linked structures such as linked lists and trees. If a pointer is a member of a structure and points to a structure of the same type, **Follow Pointer Chain** replaces the current structure with the child structure that the pointer references. For example, in Figure 10-1, hquework->begin->next is a member of the structure hquework->begin and points to another structure of type Item.
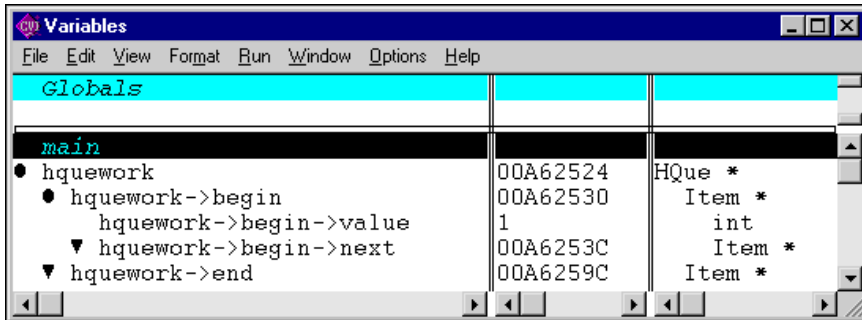
Figure 10-1. Parent Structure Pointer in a Chain

Clicking the right arrow icon or selecting **Follow Pointer Chain** replaces the current structure with the child structure that the pointer references, as shown in the Figure 10-2.
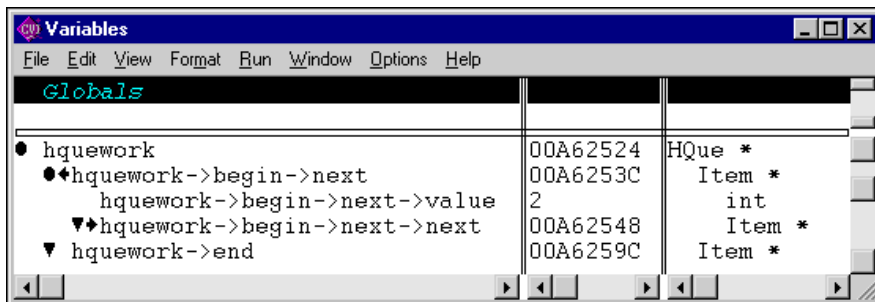
Figure 10-2. Child Structure Pointer in a Chain

View»Retrace Pointer Chain

Retrace Pointer Chain replaces the current structure with its parent. Notice the presence of the left arrow icon after selecting **Follow Pointer Chain** in Figure 10-2. This indicates that the structure hquework->begin->next is a child structure in a chain. Clicking the left arrow icon or selecting **Retrace Pointer Chain** causes the variable display to revert back, as shown in Figure 10-1.

> **Note** **Retrace Pointer Chain** is valid only when you displayed the current structure with **Follow Pointer Chain**.

View»Go to Execution Position

The **Go to Execution Position** command is available only in the Variables window when the currently highlighted item is a function name or the name of a formal parameter or local variable. The command opens the Source window that contains the call to the function in which your program suspended execution and highlights the function call. To execute the **Go to Execution Position** command, you can double-click the function name or press <Ctrl-P>, if you have default shortcut keys enabled.

View»Go to Definition

This command is valid only in the Variables window. The **Go to Definition** command opens the Source window that contains the definition of the currently selected function or variable and highlights the definition.

View»Source Code Browser

The Source Code Browser is a cross-reference tool that lists all files, functions, variables, data types, and macros in a program. You can use the browser to identify how different parts of a program interact with each other. Browse information is not available in Release configuration. For more information about the Source Code Browser, refer to the *Tools Menu for the Workspace Window* section of Chapter 2, *Workspace Window*.

View»Array Display

The **Array Display** command invokes the Array Display window for the currently highlighted array. To invoke the Array Display window, double-click an array variable or press <F4>, if you have default shortcut keys enabled.

View»String Display

The **String Display** command invokes the String Display window for the currently highlighted string. To invoke the String Display window, double-click a string variable or press <Shift-F4>, if you have default shortcut keys enabled.

View»Memory Display

The **Memory Display** command displays the currently highlighted item in the Memory Display window. If the currently highlighted item is a pointer variable, the memory pointed to by the pointer appears in the Memory Display. If the currently highlighted item is not a pointer, the address of the highlighted variable appears in the Memory Display.

View»Graphical Array View

Use the **Graphical Array View** command to view the values of arrays in a graph. You can view arrays in a graph only while you are debugging and only with 1D and 2D arrays. From the Variables window or Watch window, select **View»Graphical Array View** to open the Graphical Array View for the currently highlighted array. You also can drop a variable from the Source, Interactive Execution, Variables, or Watch window onto the Graphical Array View.

1D Arrays

The following figure shows a Graphical Array View for a 1D array.
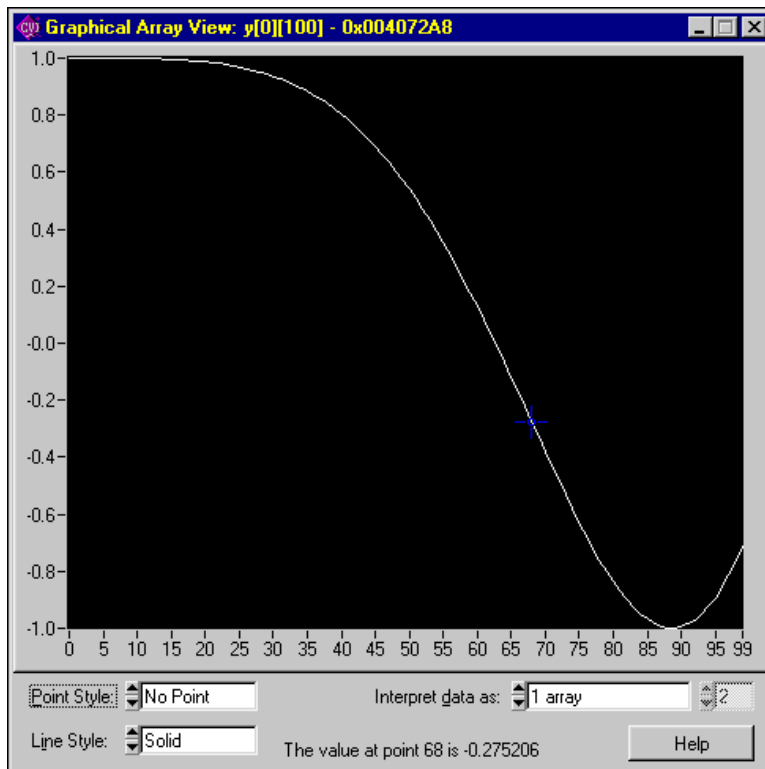
For a 1D array, the Graphical Array View shows a single plot. To find the value of a point, move the crosshair pointer over any point in the graph. When values change during debugging, the graph auto scales to fit the updated values.

You can customize the appearance of the graph with the following options:

- **Point Style**—Selects the style in which points appear.
- **Line Style**—Selects the style in which lines appear.
- **Interpret data as**—Displays the data as 1 array or as an interleaved array. Selecting **N Interleaved Arrays** displays the data as contiguous sets of points. The maximum number of points you can select is half the number of elements in your array. The following figure illustrates an interleaved 1D array.
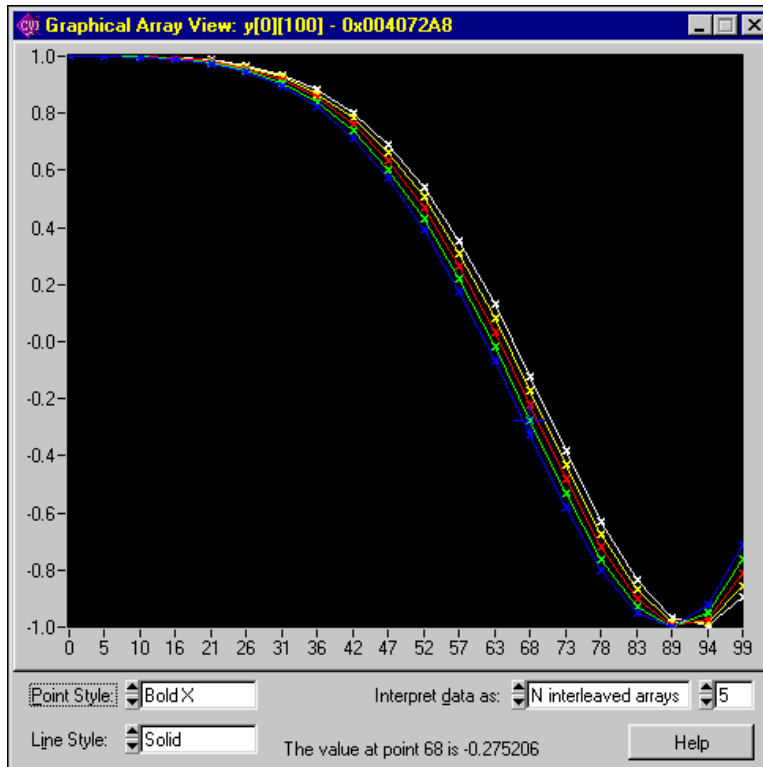
2D Arrays

The following figure shows the Graphical Array View for a 2D array.
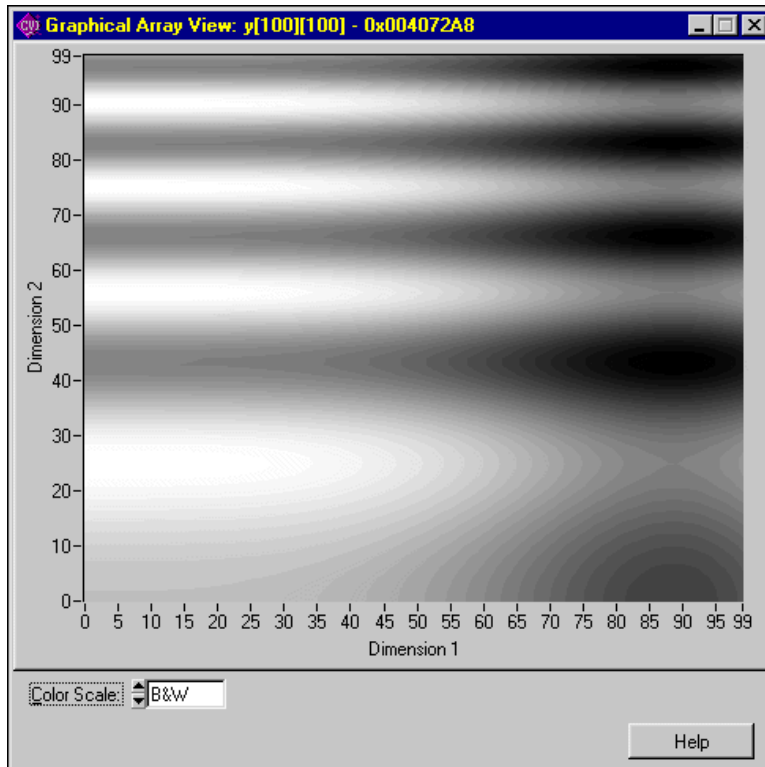
A graph for a 2D array is an intensity plot. The different shades of gray represent the magnitude of the points. Darker shades represent lower values, and lighter shades represent higher values. The color scale uses the standard spectrum. The following colors are organized from highest value to lowest value.

1. White
2. Red
3. Yellow
4. Green
5. Cyan
6. Blue
7. Magenta
8. Black

The **Dimension 1** axis represents the first dimension of the array. The **Dimension 2** axis represents the second dimension of the array.

To customize the appearance of the graph, select different color scales from the **Color Scale** option.

Format Menu for Variables and Watch Windows

Use the commands in the **Format** menu to choose the format the Variables and Watch windows use to display numbers. You can change the format for an individual variable and the default formats for all variables.

Format»Decimal/Hexadecimal/Octal/Binary/ASCII

These items in the **Format** menu specify the available formats for displaying individual integers in the Variables and Watch windows. You can display integers in decimal, hexadecimal, octal, binary, or ASCII format.

Format»Floating Point/Scientific

These items the **Format** menu specify the formats available for displaying individual real numbers. Real numbers appear in either floating-point or scientific notation.

Format»Preferences

Use the **Preferences** command to set the default formats for all integers and all real numbers.

Run Menu for Variables and Watch Windows

The commands that appear in the Variables and Watch windows **Run** menu are the same as the commands for the **Run** menu in the Workspace window. Refer to the *Run Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information about these commands.

Window Menu for Variables and Watch Windows

The commands that appear in the Variables and Watch windows **Window** menu are the same as the commands for the **Window** menu in the Workspace window. Refer to the *Window Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information about these commands.

Options Menu for Variables and Watch Windows

To use the **Options** menu commands, select a variable and then access the command from the **Options** menu.

Options»Variable Size

The **Variable Size** command displays the number of bytes the variable consumes. If you declare the variable as a buffer, the variable size is the total size of the buffer. If you declare the variable as a pointer, the **Variable Size** command displays the number of bytes the pointer itself consumes and the number of bytes in the object that the pointer references. For example, if your code contains the following declaration:

```
static double y_array [4];
```

Variable Size displays a variable size of 32 bytes for `y_array`.

Assume your code defines `dblPtr` as follows:

```
static double *dblPtr;
dblPtr = malloc (2 * sizeof(double));
```

Variable Size displays a variable size of 4 bytes for `dblPtr`, pointing to 16 bytes (2 elements).

Options»Interpret As

The **Interpret As** command displays a variable as if it were another type. Selecting a type from the Available Types dialog box displays the variable as the new type.

If **Interpret As** does not offer the exact type you want, you can use a watch expression.

Options»Estimate Number of Elements

The Variables window normally cannot expand variables for which LabWindows/CVI does not have user protection information. You can use this command to estimate the number of elements for a variable. Once you have estimated the number of elements for the variable, you can view the elements in the Variables window.

Options»Add Watch Expression

This command is not available for the Watch window. For more information about this command, refer to the *Edit Menu for Variables and Watch Windows* section.

Help Menu for Variables and Watch Windows

The commands that appear in the Variables and Watch windows **Help** menu are the same as the commands for the **Help** menu in the Workspace window. Refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information about these commands.

11

Array and String Display Windows

Array Display Window

Use the Array Display window to view and edit the contents of a string or a single-dimensional or multidimensional array during a breakpoint.

You can view the Array Display window in the following ways:

- From the Variables window, select **View»Array Display** to open the Array Display window for the currently highlighted array.
- From the Variables window, double-click an array to open the Array Display window.
- In a Source window, select **Run»View Variable Value** to open the Array Display window when the name of an array variable is under the keyboard cursor.
- In a function panel, select **Code»View Variable Value** to open the Array Display window when the name of an array variable is in the active function panel control.

You also can drop array variables onto the Array Display window. From the Source, Interactive Execution, Variables, or Watch window, drop the variable you want to view onto the Array Display window.

The following figure shows the Array Display window for a single-dimensional array.
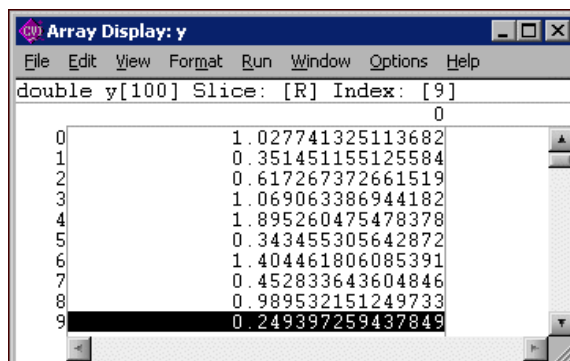
The Slice indicator shows the dimension that appears. You can display a single-dimensional array by row [R] or column [C] using the **Options»Reset Indices** command.

The Index indicator shows the currently selected element.

Multi-Dimensional Arrays

For an array with two or more dimensions, you can specify two dimensions as the rows and columns of the display. You also can specify constant values to use to fix the other dimensions. Use the **Options»Reset Indices** command to specify which plane of the array to display. The following figure shows the Array Display for a three-dimensional array.
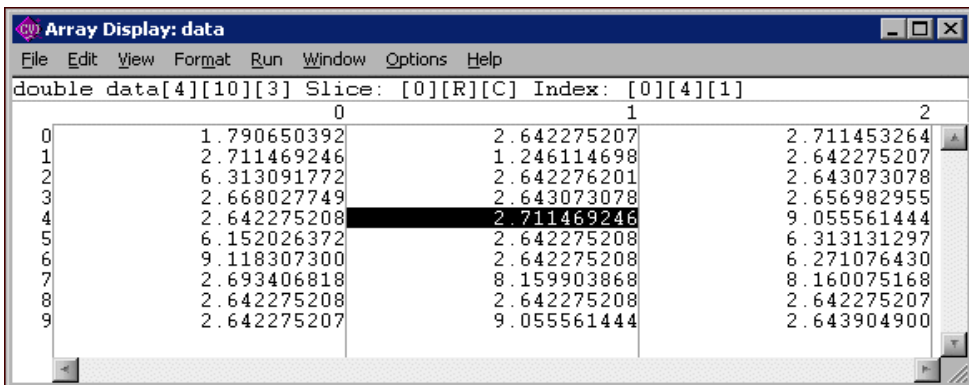
The Array Display window shows a two-dimensional view. By default, the next-to-last dimension appears as rows, the last dimension appears as columns, and the indices of the other dimensions remain constant at 0. Select **Options»Reset Indices** to specify the dimensions you want to display as rows and columns and set the other dimensions to constant values. When you select **Reset Indices** for a three-dimensional array, the Reset Indices dialog box appears.

The dialog box shows the size and display index for each array dimension. The letter R indicates the dimension displayed as rows, and the letter C indicates the dimension displayed as columns. The indices for the remaining dimensions, those dimensions not specified as either row or column, remain constant at the specified value.

If you enter an invalid character, such as a non-alphanumeric character or any alphabetic character besides R, r, C, or c, an error message appears. Likewise, if you enter an index out of the range of a dimension, an error message appears. Press <Enter> to remove the error message. If you want to close the Reset Indices dialog box without changing the indices, click **Cancel**.

String Display Window

Use the String Display window to view and edit the contents of a string variable or single-dimensional or multidimensional string array during a breakpoint.

When strings appear in ASCII format, there is no visual distinction between a space (ASCII 32) and a NUL byte (ASCII 0). You can see the difference by displaying the string in decimal format. In the String Display, select **Options»Display Entire Buffer** to see beyond the NUL byte.

You can view the String Display window in the following ways:

- From the Variables window, select **View»String Display** to open the String Display window for the currently highlighted string variable.

- From the Variables window, double-click a string to open the String Display window.

- From the Source window, select **Run»View Variable Value** to open the String Display window when the name of a string variable is under the keyboard cursor.

- From a Function Panel window, select **Code»View Variable Value** to open the String Display window when the name of a string variable is in the active function panel control.

You also can drop variables onto the String Display window. From the Source, Variables, or Watch window, select the variable you want to view and drop it onto the String Display window. You can drop only strings and string arrays into the String Display window.

Multi-Dimensional String Arrays

Use the **Reset Indices** command to specify which index of a multi-dimensional string array to use as rows in the String Display window. LabWindows/CVI disables **Reset Indices** when you view a single string variable or a one-dimensional string array. For a string array of two or more dimensions, you can specify which index to use for the rows of the display. The other dimensions remain constant at indices that you specify. When you select **Reset Indices**, the Reset Indices dialog box appears.

The dialog box shows the size and display index for each array dimension. The letter R indicates the dimension displayed as rows. The indices for the remaining dimensions remain constant at the specified values.

If you enter an invalid character or any alphabetic character besides R or r, or an invalid index, an error message appears.

File Menu for Array and String Display Windows

Many of the commands in the **File** menu work in the same way as the commands in the **File** menu for the Workspace window. Refer to the *File Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

File»Output

The **Output** command writes the contents of the window to an ASCII or binary data file on disk. When you select **Output**, a dialog box appears prompting you to specify the name of the file.

File»Input

This command is valid only in the Array Display window. Use the **Input** command to select an ASCII or binary data file on disk to replace the currently viewed array in memory.

Edit Menu for the Array and String Display Windows

The **Edit Value** and **Find** commands work in the same way as the commands in the **Edit** menu for the Variables and Watch windows. Refer to the *Edit Menu for Variables and Watch Windows* section of Chapter 10, *Variables and Watch Windows*, for more information.

Edit»Goto

The **Goto** command moves the highlight to a particular location in the current string or array plane. When you execute the **Goto** command, a dialog box appears where you can enter the row and column number of the location. For a single string, specify only the column.

Edit»Edit Character

Use the **Edit Character** command in the String Display window to change one character at a time.

Edit»Edit Mode

The **Edit Mode** command places the String Display window in edit mode so you can directly edit the string from the keyboard. This mode is valid only when you select the ASCII display format from the **Format** menu. Also, you can edit one character at a time using the **Options»Edit Character** command.

Edit»Overwrite

Use the **Overwrite** command in the String Display window to toggle between the overwrite and insert modes of editing. The **Overwrite** command has no effect unless you activate the **Edit Mode** command.

View Menu for Array and String Display Windows

The commands in the **View** menu work in the same way as the commands for the **View** menu for Variables and Watch windows. Refer to the *View Menu for the Variables and Watch Windows* section of Chapter 10, *Variables and Watch Windows*, for more information.

Format Menu for Array and String Display Windows

The following figure shows the **Format** menu.
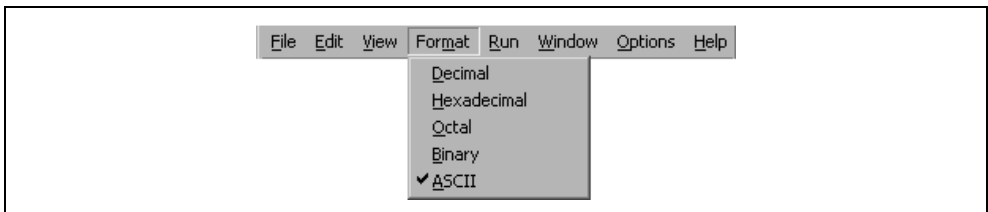
However, if a real array appears in the Array Display window, the **Format** menu appears as shown in the following figure.
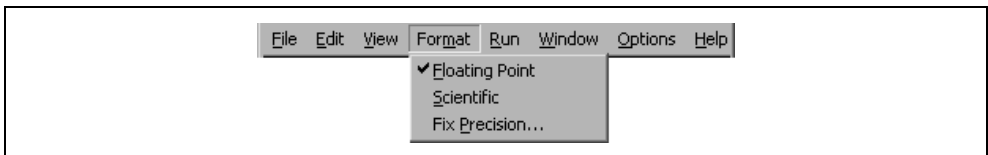
Use the commands in the **Format** menu to choose the format the Array or String Display window uses to display numbers. You can display integers in decimal, hexadecimal, octal, binary, or ASCII format. You can display real arrays in either floating-point or scientific notation.

Run Menu for Array and String Display Windows

The commands in the **Run** menu work in the same way as the commands for the **Run** menu for the Workspace window. Refer to the *Run Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Window Menu for Array and String Display Windows

The commands in the **Window** menu work in the same way as the commands for the **Window** menu for the Workspace window. Refer to the *Window Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

Options Menu for Array and String Display Windows

This section contains a detailed description of the **Options** menu for the Array and String Display windows.

Options»Reset Indices

Use **Reset Indices** in the Array Display window to set which array dimension appears as rows and which array dimension appears as columns.

Use **Reset Indices** in the String Display window to set which string array dimension appears as rows.

Options»Display Entire Buffer

This command is valid only in the String Display window. By default, the String Display window displays only the characters preceding the first ASCII NUL. To see characters beyond the NUL, select **Options»Display Entire Buffer**.

Help Menu for Array and String Display Windows

The commands in the **Help** menu work in the same way as the commands for the **Help** menu for the Workspace window. Refer to the *Help Menu for the Workspace Window* section of Chapter 2, *Workspace Window*, for more information.

A

Configuring LabWindows/CVI

This appendix describes special options that override some of the configuration defaults established during the LabWindows/CVI installation or through the configuration dialog boxes within the environment.

These options inform LabWindows/CVI where to find system files, where to place temporary files, and so on. You might not need to set any of these options.

Getting Started with LabWindows/CVI contains installation instructions for LabWindows/CVI and a hands-on tutorial. It is a good idea to be familiar with the material in *Getting Started with LabWindows/CVI* before you go through this manual.

LabWindows/CVI Startup Options

You can append certain options to the `cvi` command line, separating various parameters by spaces. The valid startup options appear in the following table.

Option	Purpose
`<filename>`	LabWindows/CVI automatically loads the file at startup. The file can be any of the types available under the **File»Open** command.
`-run`	This option automatically invokes the **Run»Debug** command.
`-run_then_exit`	This option automatically invokes **Run»Debug** and then automatically invokes **File»Exit LabWindows/CVI** when the project terminates. This option also suppresses the LabWindows/CVI startup screen.
`-newproject`	LabWindows/CVI starts with an empty Workspace window.
`-pProcessID`	LabWindows/CVI attaches to the process that *ProcessID* identifies. When the process subsequently loads DLLs, LabWindows/CVI can debug them if you created them with debugging information in LabWindows/CVI. You can express *ProcessID* as a decimal number or as a hexadecimal number that you precede with `0x`.

How to Set the Configuration Options

LabWindows/CVI development environment configuration options are under the following key, where `[version]` is the version of the LabWindows/CVI development environment:

`HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI\[version]`

Use this key to set the configuration options for the LabWindows/CVI development environment.

LabWindows/CVI Run-Time Engine configuration options are under the following key:

`HKEY_LOCAL_MACHINE\SOFTWARE\National Instruments\CVI Run-Time Engine\cvirte`

Your programs, when you run them from the environment or as stand-alone applications, use the Run-Time Engine configuration options.

A configuration string value is associated with each option, as shown in the following figure.
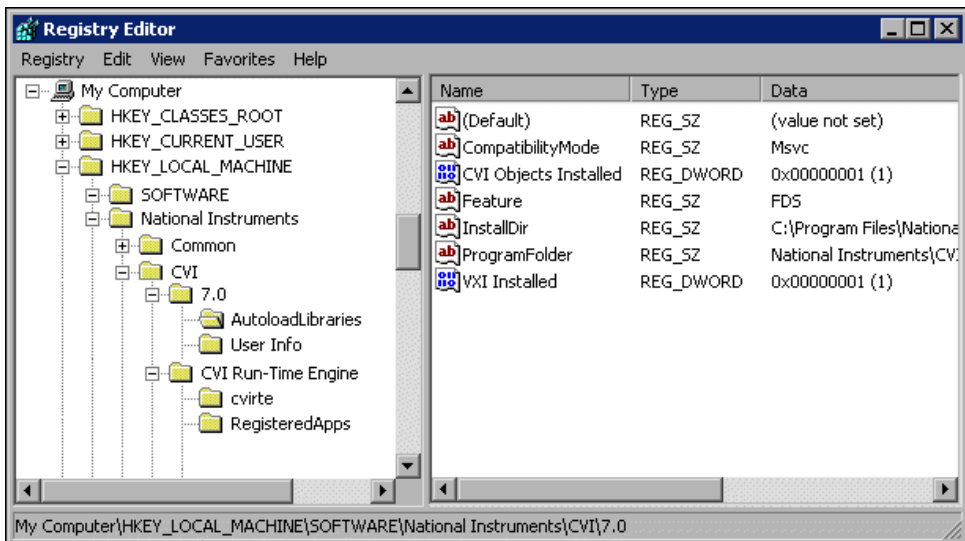
You do not have to include an unused configuration string in the Registry.

You must specify an absolute pathname, including a drive letter, for configuration strings that take a directory name.

Prompt before Shutdown Option

When you click the **X** close button of the Workspace window, LabWindows/CVI displays a message box, as shown in the following figure.
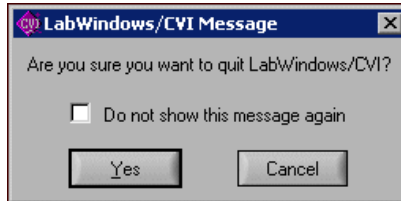
If you select **Do not show this message again** and later want to display the message box, you can make a change to the following registry key, where [version] is the version of the LabWindows/CVI environment.

```
HKEY_CURRENT_USER\Software\National Instruments\CVI\[version]\
Environment\ShowExitConfirmation
```

If you set the key to TRUE, LabWindows/CVI displays the message box. If you set the key to FALSE, LabWindows/CVI exits without displaying the message box.

Option Descriptions

When configuring LabWindows/CVI through the Registry, you can make changes to the directory options, date and time options, timer options, and font options.

Directory Options

The directory options available in LabWindows/CVI are the cvidir option and the tmpdir option.

cvidir

The `cvidir` option specifies the directory that contains the following subdirectories. You should set the `cvidir` option only if the subdirectories that LabWindows/CVI requires, shown in the following table, are not in the directory that contains the LabWindows/CVI executable.

Name of Directory	Contents
`bin`	Resource files (`cvi.rsc`, `cvimsgs.txt`), NI function panels (`.lfp` files), NI libraries (`.obj` and `.lib` files)
`fonts`	Font description files
`include`	C header files for NI libraries
`sdk`	Windows SDK

If you do not specify a directory, LabWindows/CVI assumes that the directory that contains the executable file, `cvi.exe` or `cvi`, also contains the directories in the preceding table.

tmpdir

`tmpdir` sets the location for temporary files.

If you do not specify a directory, LabWindows/CVI uses the value of the environment variable `TMP`. If the value of `TMP` is not defined or is invalid, LabWindows/CVI uses the value of the environment variable `TEMP`. If the value of `TEMP` is not defined or is invalid, LabWindows/CVI uses the directory that contains `cvi.exe`.

If you run LabWindows/CVI across a network, you must set `tmpdir` to one of your local directories.

Date and Time Option—DSTRules

Use the `DSTRules` option to specify the portions of the year in which daylight savings time is in effect in your area. This option affects ANSI C Library functions such as `mktime` and `localtime`.

For more information about `DSTRules`, refer to the *Library Reference* section of the *LabWindows/CVI Help*.

Timer Options—useDefaultTimer

You can configure LabWindows/CVI timer options in the Registry. The timer option, `useDefaultTimer`, is a DWORD value; 0 indicates `False` and 1 indicates `True`.

If you set the `useDefaultTimer` option to `True`, LabWindows/CVI uses the default Windows timer to implement the LabWindows/CVI timing-related functions, such as `Timer` and `Delay`. The default Windows timer provides a resolution of 55 ms under Windows 98 and 10 ms under Windows 2000/NT/XP/Me.

If you set `useDefaultTimer` to `False` under Windows 98, LabWindows/CVI uses the Windows multimedia library timer. The multimedia library timer provides a resolution of 1 ms.

If you set `useDefaultTimer` to `False` under Windows 2000/NT/XP/Me, LabWindows/CVI attempts to use the performance counter timer. The performance counter timer provides a resolution of 1 ms. If the performance counter timer is not available, LabWindows/CVI uses the multimedia library timer, which provides a resolution of 1 ms.

The default value for `useDefaultTimer` is `False`.

Font Options

LabWindows/CVI provides configuration options to set the fonts used throughout the environment.

DialogFontName

`DialogFontName` specifies the font that LabWindows/CVI uses in dialog boxes and the built-in pop-up panels, as shown in the following example, `DialogFontName=Courier`.

DialogFontSize

`DialogFontSize` specifies the font size that LabWindows/CVI uses in dialog boxes and the built-in pop-up panels, as shown in the following example, `DialogFontSize=30`.

DialogFontBold

`DialogFontBold` specifies whether the font that LabWindows/CVI uses in dialog boxes and the built-in pop-up panels is bold, as shown in the following example, `DialogFontBold=Yes`.

MenuFontName

`MenuFontName` specifies the font that LabWindows/CVI uses in menus, as shown in the following example, `MenuFontName=Courier`.

MenuFontSize

`MenuFontSize` specifies the font size that LabWindows/CVI uses in menus, as shown in the following example, `MenuFontSize=30`.

MenuFontBold

`MenuFontBold` specifies whether the font that LabWindows/CVI uses in menus is bold, as shown in the following example, `MenuFontBold=Yes`.

EditorFontName

`EditorFontName` specifies the default font that LabWindows/CVI uses in Source windows, as shown in the following example, `EditorFontName=Courier`.

EditorFontSize

`EditorFontSize` specifies the font size that LabWindows/CVI uses in Source windows, as shown in the following example, `EditorFontSize=30`.

EditorFontBold

`EditorFontBold` specifies whether the font that LabWindows/CVI uses in Source windows is bold, as shown in the following example, `EditorFontBold=Yes`.

MessageBoxFontName

`MessageBoxFontName` specifies the font that LabWindows/CVI uses in simple message boxes, as shown in the following example, `MessageBoxFontName=Courier`.

MessageBoxFontSize

`MessageBoxFontSize` specifies the font size that LabWindows/CVI uses in simple message boxes, as shown in the following example, `MessageBoxFontSize=30`.

MessageBoxFontBold

`MessageBoxFontBold` specifies whether the font that LabWindows/CVI uses in simple message boxes is bold, as shown in the following example, `MessageBoxFontBold=Yes`.

AppFontName

`AppFontName` specifies the font that LabWindows/CVI uses for dialog box labels and for function panels, as shown in the following example, `AppFontName=Courier`.

AppFontSize

`AppFontSize` specifies the font size that LabWindows/CVI uses in dialog box labels and for function panels, as shown in the following example, `AppFontSize=30`.

AppFontBold

`AppFontBold` specifies whether the font that LabWindows/CVI uses in dialog box labels and for function panels is bold, as shown in the following example, `AppFontBold=Yes`.

B

Technical Support and Professional Services

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

- **Support**—Online technical support resources include the following:

  - **Self-Help Resources**—For immediate answers and solutions, visit our extensive library of technical support resources available in English, Japanese, and Spanish at `ni.com/support`. These resources are available for most products at no cost to registered users and include software drivers and updates, a KnowledgeBase, product manuals, step-by-step troubleshooting wizards, conformity documentation, example code, tutorials and application notes, instrument drivers, discussion forums, a measurement glossary, and so on.

  - **Assisted Support Options**—Contact NI engineers and other measurement and automation professionals by visiting `ni.com/support`. Our online system helps you define your question and connects you to the experts by phone, discussion forum, or email.

- **Training**—Visit `ni.com/training` for self-paced tutorials, videos, and interactive CDs. You also can register for instructor-led, hands-on courses at locations around the world.

- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

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

Glossary

A

active window	The window that user input affects at a given moment. The title of an active window is highlighted.
Array Display	A window for viewing and editing numeric arrays.
auto-exclusion	A mechanism that prevents existing lines from executing in the Interactive Execution window.

B

binary control	A function panel control that resembles a physical on/off switch and can produce one of two values depending on the position of the switch.
breakpoint	An interruption in the execution of a program.
button	A dialog box item that executes a command associated with the dialog box.

C

checkbox	A dialog box item that allows you to toggle between two possible options.
click	A mouse-specific term; to quickly press and release the mouse button.
Clipboard	A temporary storage area LabWindows/CVI uses to hold text that is cut or copied from a work area.
CodeBuilder	The LabWindows/CVI feature that creates code based on a `.uir` file to connect your GUI to the rest of your program. This code can be compiled and run as soon as it is created.
common control	A control on a common control function panel that specifies a parameter in all functions associated with a Function Panel window.
compiler define	A command-line argument passed to the compiler that defines an identifier as a macro to the preprocessor.

control	An input and output device that appears on a function panel for specifying function parameters and displaying function results.
cursor	The flashing rectangle that shows where you can enter text on the screen. If you have a mouse installed, there is also a mouse cursor.
cursor location indicator	An element of the LabWindows/CVI screen that specifies the row and column position of the cursor in the window.

D

default command	The action that takes place when <Enter> is pressed and no command is specifically selected. Default command buttons are indicated in dialog boxes with an outline.
dialog box	A prompt mechanism in which you specify additional information needed to complete a command.
double-click	A mouse-specific term; to click the mouse button twice in rapid succession.
drag	A mouse-specific term; to hold down the mouse button while moving the mouse across a flat surface, such as a mouse pad.

E

entry mode indicator	An element of the LabWindows/CVI screen that indicates the current text mode as either insert or overwrite.
excluded code	Code that is ignored during compilation and execution. Excluded lines of code are displayed in a different color than included lines of code.

F

.fp file	A file containing information about the function tree and function panels of an instrument module.
function panel	A screen-oriented user interface to the LabWindows/CVI libraries in which you can interactively execute library functions and generate code for inclusion in a program.

Function Panel Editor	The window in which you build a function panel.
Function Panel window	The window that contains function panels.
function tree	The hierarchical structure in which the functions in a library or an instrument driver are grouped. The function tree simplifies access to a library or instrument driver by presenting functions organized according to the operation they perform, as opposed to a single linear listing of all available functions.
Function Tree Editor	The window in which you build the skeleton of a function panel file.

G

generated code box	A text box located at the bottom of the function panel window that displays the function call that corresponds to the current state of the function panel controls.
global control	A function panel control that displays the contents of global variables in a library function. Global controls allow you to monitor global variables in a function that the function does not specifically return as results. These are read-only controls that the user cannot alter and do not contribute a parameter to the generated code.
Graphical Array Display	A window in which you can view the values of arrays in a graph.

H

hex	Hexadecimal.
highlight	The way in which input focus is displayed on a LabWindows/CVI screen; to move the input focus onto an item.

I

immediate action menu	A menu that has no menu items associated with it and causes a command to execute immediately. An immediate action command is suffixed with an exclamation point (`!`).

input control	A function panel control that accepts a value typed in from the keyboard. An input control can have a default value associated with it. This value appears in the control when the panel is first displayed.
input focus	Displayed on the screen as a highlight on an item, signifying that the item is active. User input affects the item in the dialog box that has the input focus.
instrument driver	A set of high-level functions for controlling an instrument. It encapsulates many low-level operations, such as data formatting and GPIB, RS-232, and VXI communication, into intuitive, high-level functions. An instrument driver can pertain to one particular instrument or to a group of related instruments. An instrument driver consists of a program and a set of function panels. The program contains the code for the high-level functions. Associated with the instrument program is an include file that declares the high-level functions you can call, the global variables you can access, and the defined constants you can use.
Interactive Execution window	A LabWindows/CVI work area in which sections of code may be executed without creating an entire program.

L

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

M

MB	Megabytes of memory.
menu	An area accessible from a menu bar that displays selectable menu items.

N

new style (function definition)	A function definition in which parameters are declared directly in the parameter list.

O

old style (function definition)	A function definition in which parameters are declared outside of the parameter list.
output control	A function panel control that displays a value that the function you execute generates. An output control parameter must be a string, an array, or a reference parameter of type integer, long, single-precision, or double-precision.
Output Window Region	An area of the Workspace window in which errors, output, and search match windows appear.

P

project	A list of files, usually including a source file, user interface resource file, and header file, that your application uses.
Project Tree	An area of the Workspace window that contains the lists of projects and files in the current workspace.
prompt command	A command that requires additional information before it can be executed; a prompt command appears on a pull-down menu suffixed with three ellipses (…).

R

return value control	A function panel control that displays a value returned from a function as a return value rather than as a formal parameter.
ring control	A function panel control that represents a range of values much like the slide control but displays only a single item in a list rather than displaying the whole list at once as the slide control does. Each item has a different value associated with it. This value is placed in the function call.

S

scroll bars	Areas along the bottom and right sides of a window that show your relative position in the file. Scroll bars can be used with a mouse to move about in the window.

scrollable text box	A dialog box item that displays text in a scrollable display.
select	To choose the item that the next executed action will affect by moving the input focus (highlight) to a particular item or area.
shortcut key commands	A combination of keystrokes that provides a means of executing a command without accessing a menu in the menu bar.
slide control	A function panel control that resembles a physical slide switch. A slide control is a means for selecting one item from a list of options; it inserts a value in a function call that depends on the position of the crossbar on the switch.
slider	The crossbar on the slide control that determines the value placed in the function call.
Source window	A LabWindows/CVI work area in which programs are edited and executed.
Standard Input/Output window	A LabWindows/CVI work area in which textual output to and input from the user take place.
standard libraries	The LabWindows/CVI User Interface, ActiveX, DataSocket, Advanced Analysis (or Analysis), Formatting and I/O, GPIB/GPIB 488.2, Utility, RS-232, TCP Support, DDE Support, VISA, IVI, Traditional NI-DAQ, NI-DAQmx, and ANSI C libraries.
step mode	A program execution mode in which a program is manually executed one instruction at a time. Each instruction in the program is highlighted as it is executed.
String Display window	A window for viewing and editing string variables and arrays.

T

text box	A dialog box item in which text is entered from the keyboard.
timer control	A user interface control that schedules the periodic execution of a callback function. A typical use of this control might be to update a graph every second.
Tooltips	A small, yellow box that displays the value of variables and expressions in a Source window.

U

User Interface Editor	The window in which you build pull-down menus, dialog boxes, panels, and controls and save them to a user interface resource (`.uir`) file.

V

Variables window	A window that shows the values of all the currently active variables.

W

Watch window	A window that shows the values of user-selectable variables and expressions that are currently active.
window	A working area that supports specific tasks related to developing and executing programs.
Window Confinement Region	An area of the Workspace window that contains open Source, User Interface Editor, and Function Tree Editor windows.
work area	The area of the LabWindows/CVI screen that contains the text displayed in a window.
workspace	A file that contains settings that do not affect the way a project builds, such as breakpoints, window position, tag information, and debugging levels. A workspace can contain more than one project.
Workspace window	The main work area in LabWindows/CVI; contains the Project Tree, Library Tree, Window Confinement Region, and Output Window Region.

Index

A

About LabWindows/CVI command, 2-73

ActiveX Container Support option, Create Distribution Kit dialog box, 2-31

Add Files to Executable button, Target Settings dialog box, 2-17

Add Files to Project, 2-8, 2-9, 3-4

Add Missing Includes command, 4-21

Add Program File to Project command, 6-6, 7-1, 8-2, 9-4

Add to Source Control command, 2-53

Add Watch Expression command, 4-25, 6-5, 6-11, 10-2, 10-5, 10-13

Add/Edit Tools Menu Item dialog box, 2-56

Add/Edit Watch Expression dialog box, 10-2

Align command, 3-16

Alignment command, 3-15

All Callbacks command, 3-18

All Code command, 3-18, 3-19

All Files command, Add Files to Project dialog box, 2-9

Alphabetize option, Select Function Panel dialog box, 2-45

Always Append Code to End option, Preferences command, 3-19

ANSI C Library display, External Compiler Support dialog box, 2-25

Any Array data type, 5-9

Any Type data type, 5-9

AppFontBold configuration option, A-7

AppFontName configuration option, A-7

AppFontSize configuration option, A-7

Application File option, Target Settings dialog box, 2-14

Application Icon File option, Target Settings dialog box, 2-14

Application Title option, Target Settings dialog box, 2-14

applications, creating, 1-3

Apply Default Font command, 3-11

Arrange menu, User Interface Editor

  Align command, 3-16

  Alignment command, 3-15

  Center Label command, 3-17

  Control Coordinates command, 3-17

  Control ZPlane Order command, 3-17

  Distribute command, 3-17

  Distribution command, 3-16

array data types, user-defined, 5-11

Array Display command, 10-8

Array Display command, View menu, 11-1

Array Display window

  Edit menu, 11-4

  File menu, 11-4

  Format menu, 11-5

  Help menu, 11-6

  multi-dimensional array (figure), 11-2

  Options menu, 11-6

  purpose and use, 1-3, 11-1

  Run menu, 11-5

  single-dimensional array (figure), 11-1

  View menu, 11-5

  viewing, 11-1

  Window menu, 11-6

arrays

  multi-dimensional arrays

    illustration, 11-2

    Reset Indices dialog box, 11-2

    specifying dimensions, 11-2

  one-dimensional array

    displaying in Array Display window (figure), 11-1

    Graphical Array View (figures), 10-9

  two-dimensional array, Graphical Array View (figure), 10-11

F

M

P