# LabWindows/CVI

## LabWindows/CVI
## User Manual

NATIONAL INSTRUMENTS™
*The Software is the Instrument™*

February 1998 Edition
Part Number 320681D-01

**Internet Support**
E-mail: support@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

**Bulletin Board Support**
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

**Fax-on-Demand Support**
512 418 1111

**Telephone Support (USA)**
Tel: 512 795 8248
Fax: 512 794 5678

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

**National Instruments Corporate Headquarters**
6504 Bridge Point Parkway  Austin, Texas 78730-5039  USA  Tel: 512 794 0100

© Copyright 1994, 1998 National Instruments Corporation. All rights reserved.

# Important Information

## Warranty

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

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

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

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

## Copyright

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

## Trademarks

CVI™, `natinst.com`™, National Instruments™, the National Instruments logo, and The Software is the Instrument™are trademarks of National Instruments Corporation.

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

## WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

# Contents

## About This Manual


Organization of This Manual ..... xxi
Conventions Used in This Manual ..... xxii
LabWindows/CVI Documentation Set ..... xxiii
Related Documentation ..... xxiv
Customer Communication ..... xxiv


## Chapter 1
## Configuring LabWindows/CVI


LabWindows/CVI Startup Options ..... 1-1
How to Set the Configuration Options ..... 1-2
Windows 95/NT ..... 1-2
Windows 3.1 ..... 1-3
UNIX ..... 1-3
Cross-Platform Option Descriptions ..... 1-4
Directory Options ..... 1-4
cfgdir ..... 1-4
Using cfgdir under Windows 3.1 ..... 1-4
Using cfgdir under UNIX ..... 1-4
Environment Settings Stored in Registry under Windows 95/NT ..... 1-4
cvidir ..... 1-4
resdir [obsolete] ..... 1-5
tmpdir ..... 1-5
Using tmpdir in Windows ..... 1-5
Using tmpdir under UNIX ..... 1-5
Date and Time Options: DSTRules ..... 1-6
Windows Option Descriptions ..... 1-6
Timer Options: useDefaultTimer ..... 1-6
Font Options ..... 1-6
DialogFontName ..... 1-6
DialogFontSize ..... 1-7
DialogFontBold ..... 1-7
Debug Options (Windows 3.1 Only) ..... 1-7
DisplayCVIDebugVxDMissingMessage ..... 1-7
CatchProtectionFaults ..... 1-7
LoadCVIDebugVxD ..... 1-7

UNIX Option Descriptions ..... 1-8
Font Options ..... 1-8
dialogFont ..... 1-8
editorFont ..... 1-8
menuFont ..... 1-8
appFont ..... 1-8
messageBoxFont ..... 1-8
Display, Mouse, and Keyboard Options ..... 1-8
activate ..... 1-8
useDefaultColors ..... 1-9
useMetaKey ..... 1-9
warpMouseOverDialogBoxes ..... 1-9

## Chapter 2
## LabWindows/CVI Overview

Components of LabWindows/CVI ..... 2-1
Standard Libraries ..... 2-2
User Interface Library ..... 2-3
Data Acquisition Library and Easy I/O for DAQ Library ..... 2-3
VISA Library and IVI Library ..... 2-3
Instrument Library ..... 2-3
LabWindows/CVI Environment ..... 2-4
How to Create Applications with LabWindows/CVI ..... 2-5
Creating a User Interface ..... 2-6
Creating Standalone Programs and DLLs ..... 2-6

## Chapter 3
## Project Window

Project Window Overview ..... 3-1
File Menu ..... 3-3
New ..... 3-4
Open ..... 3-4
Save ..... 3-5
Save As. . . ..... 3-5
Save All ..... 3-5
Auto Save Project ..... 3-5
Print ..... 3-6
Most Recently Closed Files ..... 3-6
Exit LabWindows/CVI ..... 3-6
Edit Menu ..... 3-6
Add Files to Project. . . ..... 3-7
Exclude File from Build/Include File in Build ..... 3-8

Remove File....3-8
Move Item Up....3-8
Move Item Down....3-8
View Menu....3-8
Show Full Path Names....3-8
Show Full Dates....3-9
Sort By Date....3-9
Sort By Name....3-9
Sort By Pathname....3-9
Sort By File Extension....3-9
No Sorting....3-9
Build Menu....3-10
Compile File....3-10
Build Project....3-10
Link Project....3-11
Update Program Files from Disk....3-11
Mark File for Compilation....3-11
Mark All for Compilation....3-11
Target (Windows 95/NT Only)....3-11
Instrument Driver Support Only....3-12
Utility Library Functions....3-13
No Standard Input/Output....3-13
Multithread Safe....3-13
Generate Makefile (UNIX Only)....3-13
Create Standalone Executable. . .....3-13
Create Dynamic Link Library (Windows 95/NT Only)....3-15
Create Static Library (Windows 95/NT Only)....3-18
DLL Debugging (Windows 95/NT Only)....3-19
Location of Files Required for Debugging DLLs....3-19
Different Ways to Debug DLLs....3-20
Running a Program in LabWindows/CVI....3-20
Running an External Process....3-21
Multithreaded Executables 3-21
External Compiler Support (Windows 95/NT Only)....3-21
Create Distribution Kit (Windows Only)....3-24
Build Information Section....3-25
File Groups Section....3-27
Main Section....3-28
Advanced Distribution Kit Options....3-29
Installation Script File Section....3-29
Executable to Run After Setup....3-30
Installation Titles....3-30

Run Menu ..... 3-30
Run Project ..... 3-31
Run-Time Error Reporting ..... 3-31
Continue ..... 3-31
Terminate Execution ..... 3-31
Break at First Statement ..... 3-31
Breakpoints. . . ..... 3-31
Select External Process. . . ..... 3-32
Execute ..... 3-32
Using Instrument Drivers ..... 3-32
Instrument Driver Files ..... 3-33
VXIplug&play Instrument Driver Files ..... 3-33
Loading/Unloading Instrument Drivers ..... 3-34
Precedence Rules for Loading the Instrument Driver Program File .. 3-34
Loading an Instrument without an Instrument Program ..... 3-36
Modules That Contain Non-Instrument Functions ..... 3-36
Modifying an Instrument Driver ..... 3-36
Instrument Menu ..... 3-37
Load. . . ..... 3-37
File Format Conversion ..... 3-38
Unload. . . ..... 3-38
Edit. . . ..... 3-38
Accessing Function Panels from the Instrument Menu ..... 3-40
Library Menu ..... 3-41
User Interface. . . ..... 3-42
Analysis. . ./Advanced Analysis. . . ..... 3-42
Easy I/O For DAQ. . . (Windows Only) ..... 3-43
Data Acquisition. . . (Windows Only) ..... 3-43
VXI. . . ..... 3-43
GPIB/GPIB 488.2. . . ..... 3-43
RS-232. . . ..... 3-43
VISA. . . ..... 3-44
IVI. . . ..... 3-44
TCP. . . ..... 3-44
X Property. . . (UNIX Only) ..... 3-44
DDE. . . (Windows Only) ..... 3-44
ActiveX Automation. . . (Windows 95/NT Only) ..... 3-44
Formatting and I/O. . . ..... 3-45
Utility. . . ..... 3-45
ANSI C. . . ..... 3-45
User Libraries ..... 3-45
Dummy .fp Files for Support Libraries ..... 3-45
System Libraries ..... 3-46

Tools Menu ..... 3-46
Create ActiveX Automation Controller. . . (Windows 95/NT Only) ..... 3-46
Browse Type Library Dialog Box ..... 3-47
Updating Existing Controller Instrument Drivers ..... 3-52
Create IVI Instrument Driver. . . ..... 3-52
User-Defined Entries in the Tools Menu ..... 3-52
Window Menu ..... 3-53
Cascade Windows ..... 3-53
Tile Windows ..... 3-53
Minimize All (Windows 95/NT Only) ..... 3-53
Close All ..... 3-53
Project ..... 3-54
Build Errors ..... 3-54
Run-Time Errors ..... 3-54
Variables ..... 3-54
Watch ..... 3-54
Array/String Displays ..... 3-54
User Interface ..... 3-55
Function Panel ..... 3-55
Function Tree ..... 3-55
Help Editor ..... 3-55
Interactive Execution ..... 3-55
Standard Input/Output ..... 3-56
Open Source Files ..... 3-56
Options Menu ..... 3-56
Compiler Options. . . ..... 3-57
Compiler Defines. . . ..... 3-59
Include Paths. . . ..... 3-61
Instrument Directories. . . ..... 3-61
Run Options. . . ..... 3-62
Command Line. . . ..... 3-63
Environment. . . ..... 3-63
Keyboard Options. . . (SPARCstation Only) ..... 3-64
Library Options. . . ..... 3-64
User Libraries ..... 3-65
Dummy .fp Files for Support Libraries ..... 3-66
National Instruments Libraries ..... 3-66
Tools Menu Options. . . ..... 3-67
Project Move Options. . . ..... 3-68
Font. . ..... 3-69
Colors. . . ..... 3-69

# Chapter 4
# Source, Interactive Execution, and Standard Input/Output Windows


Source Windows ..... 4-1
Toolbars in LabWindows/CVI ..... 4-1
Modifying Your Toolbars ..... 4-2
Positioning Buttons and Separators on the Toolbar ..... 4-2
Adding and Positioning Buttons ..... 4-2
Adding and Positioning Separators ..... 4-3
Other Positioning Controls ..... 4-3
Notification of External Modification (Windows Only) ..... 4-3
Context Menus ..... 4-3
Interactive Execution Window ..... 4-3
Standard Input/Output Window ..... 4-5
Using Subwindows ..... 4-5
Selecting Text in the Source and Interactive Execution Windows ..... 4-5
File Menu ..... 4-7
New ..... 4-7
Open ..... 4-8
Open Quoted Text ..... 4-8
Save ..... 4-8
Save As. . . ..... 4-8
Save Copy As. . . ..... 4-8
Close ..... 4-8
Save All ..... 4-8
Add File to Project ..... 4-8
Read Only ..... 4-9
Print. . . ..... 4-9
Exit LabWindows/CVI ..... 4-9
Edit Menu ..... 4-9
Undo and Redo ..... 4-10
Cut and Copy ..... 4-11
Paste ..... 4-11
Delete ..... 4-11
Select All ..... 4-11
Clear Window ..... 4-11
Toggle Exclusion ..... 4-12
Resolve All Excluded Lines ..... 4-12
Insert Construct ..... 4-12
Balance ..... 4-12
Diff ..... 4-13
Go To Definition ..... 4-13
Find. . . ..... 4-14

Replace. . . 4-17
Next File 4-18
View Menu 4-19
Line Numbers 4-19
Line Icons 4-19
Toolbar 4-20
Line. . . 4-20
Beginning/End of Selection 4-20
Toggle Tag 4-20
Next Tag 4-20
Previous Tag 4-20
Tag Scope 4-20
Clear Tags. . . 4-20
Function Panel History. . . 4-21
Function Panel Tree 4-21
Recall Function Panel 4-21
Invoking the Recall Function Panel Command 4-21
Recalling a Function Panel from a Function Name Only 4-21
Multiple Panels for One Function 4-22
Multiple Functions in One Function Panel Window 4-22
Syntax Requirements for the Recall Function Panel Command 4-22
Find Function Panel. . . 4-22
Find UI Object 4-23
Build Menu 4-23
Compile File 4-23
Build Project 4-24
Link Project 4-24
Mark File for Compilation 4-24
Clear Interactive Declarations 4-24
Insert Include Statements. . . 4-24
Add Missing Includes 4-25
Generate Prototypes 4-25
Next/Previous Build Error 4-25
Build Errors in Next File 4-25
Run Menu 4-26
Introduction to Breakpoints and Watch Expressions 4-26
Breakpoint State 4-27
Setting and Clearing Breakpoints 4-27
Conditional Breakpoints 4-28
Watch Expressions 4-28

Run Project/Run Interactive Statements ..... 4-28
Running in a Source Window ..... 4-28
Running in the Interactive Execution Window ..... 4-28
Run-Time Error Reporting ..... 4-29
Continue ..... 4-29
Go To Cursor ..... 4-29
Step Over ..... 4-29
Step Into ..... 4-30
Finish Function ..... 4-30
Terminate Execution ..... 4-30
Close Libraries ..... 4-30
Break at First Statement ..... 4-30
Toggle Breakpoint ..... 4-30
Breakpoints. . . ..... 4-31
Activate Panels when Resuming ..... 4-32
Stack Trace. . . ..... 4-33
Up Call Stack ..... 4-33
Down Call Stack ..... 4-33
View Variable Value ..... 4-33
Add Watch Expression ..... 4-33
Dynamic Memory. . . ..... 4-33
Instrument Menu ..... 4-34
Library Menu ..... 4-34
Tools Menu ..... 4-34
Create ActiveX Automation Controller. . . (Windows 95/NT Only) ..... 4-34
Update ActiveX Automation Controller. . . (Windows 95/NT Only) ..... 4-34
Create IVI Instrument Driver. . . ..... 4-35
Edit Instrument Attributes. . . ..... 4-35
Edit Function Tree ..... 4-35
Edit Function Panel ..... 4-36
Window Menu ..... 4-36
Options Menu ..... 4-36
Editor Preferences. . . ..... 4-37
Undo ..... 4-37
Paste ..... 4-37
Tabs ..... 4-38
Line Terminator ..... 4-38
Toolbar. . . ..... 4-38
Bracket Styles. . . ..... 4-38
Font. . . ..... 4-38
Colors. . . ..... 4-38
Syntax Coloring ..... 4-39
User Defined Tokens for Coloring. . . ..... 4-39

Translate DOS LW Program. . . .....4-39
Generate DLL Import Source. . . (Windows 95/NT Only) .....4-39
Generate DLL Import Library. . . (Windows 95/NT Only).....4-40
Generate DLL Glue Source. . . (Windows 3.1 Only).....4-40
Generate DLL Glue Object. . . (Windows 3.1 Only) .....4-41
Generate Visual Basic Include. . . (Windows Only) .....4-41
Create Object File. . . .....4-41
Help Menu .....4-42

## Chapter 5
## Using Function Panels

Accessing Function Panels .....5-1
Multiple Function Panels in a Window .....5-3
Generated Code Box.....5-3
Toolbars in LabWindows/CVI.....5-3
Function Panel Controls.....5-4
Specifying a Return Value Control Parameter .....5-4
Specifying an Input Control Parameter .....5-5
Specifying a Numeric Control Parameter.....5-5
Specifying a Slide Control Parameter .....5-5
Specifying a Binary Control Parameter.....5-6
Specifying a Ring Control Parameter.....5-6
Specifying an Output Control Parameter .....5-6
Using a Global Control.....5-7
Common Control Function Panel.....5-7
Convenient Viewing of Function Panel Variables .....5-7
File Menu.....5-7
New.....5-7
Open .....5-8
Close.....5-8
Save All .....5-8
Add .FP File to Project .....5-8
Add Program File to Project.....5-8
Exit LabWindows/CVI.....5-8
Code Menu.....5-9
Run Function Panel .....5-9
Declare Variable. . . .....5-10
Clear Interactive Declarations .....5-11
Close Libraries.....5-11
Select UIR Constant. . . .....5-11
Selecting Constants from .uir Files .....5-12

Select Attribute Constant. . . 5-13
Selecting Constants in an Attribute Control 5-13
Selecting Constants in a Value Control 5-14
Select Variable. . . 5-15
What Is Included in the List Box 5-17
Data Type Compatibility 5-17
Sorting of List Box Entries 5-18
Insert Function Call 5-18
Set Target File. . . 5-18
View Variable Value 5-19
Add Watch Expression 5-19
View Menu 5-19
Toolbar 5-19
Error 5-19
Include File 5-20
Current Tree. . . 5-20
Function Panel History. . . 5-20
Find Function Panel. . . 5-20
Previous Function Panel 5-20
Next Function Panel Window 5-20
Previous Function Panel Window 5-21
Next Function Panel Window 5-21
First Function Panel Window 5-21
Last Function Panel Window 5-21
Instrument Menu 5-21
Library Menu 5-21
Window Menu 5-21
Options Menu 5-22
Default Control 5-22
Default All 5-22
Toolbar. . . 5-22
Exclude Function 5-22
Toggle Control Style 5-22
Change Format. . . 5-23
Edit Function Panel Window 5-23
Help Menu 5-23
Control 5-24
Function 5-24

# Chapter 6
# Variables and Watch Windows


Variables Window ..... 6-1
Watch Window ..... 6-3
File Menu ..... 6-5
- New ..... 6-6
- Open ..... 6-6
- Output. . . ..... 6-6
- Hide ..... 6-6
- Save All ..... 6-6
- Exit LabWindows/CVI ..... 6-6

Edit Menu for the Variables Window ..... 6-6
- Edit Value. . . ..... 6-7
- Find. . . ..... 6-7
- Next Scope ..... 6-8
- Previous Scope ..... 6-8

Edit Menu for the Watch Window ..... 6-9
- Edit Value. . . ..... 6-9
- Add Watch Expression. . . ..... 6-9
- Edit Watch Expression. . . ..... 6-9
- Delete Watch Expression ..... 6-9
- Find. . . ..... 6-9

View Menu ..... 6-10
- Expand Variable ..... 6-10
- Close Variable ..... 6-11
- Follow Pointer Chain ..... 6-11
- Retrace Pointer Chain ..... 6-12
- Go To Execution Position ..... 6-12
- Go To Definition ..... 6-12
- Array Display ..... 6-13
- String Display ..... 6-13

Format Menu ..... 6-13
Run Menu ..... 6-14
Window Menu ..... 6-14
Options Menu ..... 6-14
- Variable Size. . . ..... 6-15
- Interpret As. . . ..... 6-15
- Estimate Number of Elements. . . ..... 6-15
- Add Watch Expression. . . ..... 6-15

## Chapter 7
## Array and String Display Windows

Array Display Window ..... 7-1
Multi-Dimensional Arrays ..... 7-2
String Display Window ..... 7-4
Multi-Dimensional String Array ..... 7-4
File Menu ..... 7-5
New ..... 7-5
Open ..... 7-5
Output ..... 7-5
Input (Array Display Only) ..... 7-5
Close ..... 7-6
Save All ..... 7-6
Exit LabWindows/CVI ..... 7-6
Edit Menu for the Array Display Window ..... 7-6
Edit Value. . . ..... 7-6
Find. . . ..... 7-6
Goto. . . ..... 7-7
Edit Menu for the String Display Window ..... 7-7
Edit Character. . . ..... 7-7
Edit Mode ..... 7-7
Overwrite ..... 7-7
Find. . . ..... 7-8
Goto. . . ..... 7-8
Format Menu ..... 7-8
Run Menu ..... 7-9
Window Menu ..... 7-9
Options Menu ..... 7-9
Reset Indices. . . ..... 7-9
Display Entire Buffer (String Display Only) ..... 7-10

## Appendix A
## Source Window Keyboard Commands

## Appendix B
## Customer Communication

## Glossary

## Index

## Figures

Figure 1-1. Registry for Windows 95 ..... 1-2

Figure 2-1. Project Window ..... 2-5

Figure 3-1. Project Window ..... 3-2
Figure 3-2. File Menu ..... 3-3
Figure 3-3. New Command Submenu ..... 3-4
Figure 3-4. Open Command ..... 3-4
Figure 3-5. Edit Menu ..... 3-6
Figure 3-6. Add Files to Project Command Submenu ..... 3-7
Figure 3-7. View Menu ..... 3-8
Figure 3-8. Build Menu ..... 3-10
Figure 3-9. Create Standalone Executable Dialog Box ..... 3-14
Figure 3-10. Create Dynamic Link Library Dialog Box ..... 3-15
Figure 3-11. Create Static Library Dialog Box ..... 3-18
Figure 3-12. External Compiler Support Dialog Box ..... 3-22
Figure 3-13. Create Distribution Kit Dialog Box ..... 3-25
Figure 3-14. Advanced Distribution Kit Options Dialog Box ..... 3-29
Figure 3-15. Run Menu ..... 3-30
Figure 3-16. Instrument Menu ..... 3-32
Figure 3-17. Instrument Menu with Two Instruments Loaded ..... 3-37
Figure 3-18. Edit Instrument Dialog Box ..... 3-38
Figure 3-19. Instrument Driver Dialog Box ..... 3-39
Figure 3-20. Select Function Panel Dialog Box ..... 3-40
Figure 3-21. Instrument Help Dialog Box ..... 3-41
Figure 3-22. Library Menu ..... 3-42
Figure 3-23. Tools Menu ..... 3-46
Figure 3-24. Select ActiveX Automation Server Dialog Box ..... 3-47
Figure 3-25. Browse Type Library Dialog Box for OfficeCompatible 1.0 ..... 3-49
Figure 3-26. Select Target File Dialog Box ..... 3-51
Figure 3-27. Window Menu ..... 3-53
Figure 3-28. Options Menu ..... 3-56
Figure 3-29. Library Options Dialog Box ..... 3-65
Figure 3-30. Tools Menu Options Dialog Box ..... 3-67
Figure 3-31. Add/Edit Tools Menu Item Dialog Box ..... 3-68

Figure 4-1. Customize Source Window Toolbar Dialog Box ..... 4-2
Figure 4-2. Selecting Text Using Character Select Mode ..... 4-6
Figure 4-3. Selecting Text Using Line Select Mode ..... 4-6
Figure 4-4. Selecting Text Using Column Select Mode ..... 4-7
Figure 4-5. File Menu ..... 4-7
Figure 4-6. Edit Menu ..... 4-10
Figure 4-7. Diff Submenu ..... 4-13
Figure 4-8. Find Dialog Box ..... 4-14
Figure 4-9. Find Button Bar ..... 4-17
Figure 4-10. Replace Button Bar ..... 4-17
Figure 4-11. View Menu ..... 4-19
Figure 4-12. Build Menu ..... 4-23
Figure 4-13. Run Menu ..... 4-26
Figure 4-14. Breakpoints Dialog Box ..... 4-31
Figure 4-15. Edit Breakpoint Dialog Box ..... 4-31
Figure 4-16. Tools Menu ..... 4-34
Figure 4-17. Options Menu ..... 4-36
Figure 4-18. Editor Preferences ..... 4-37
Figure 4-19. Help Menu ..... 4-42

Figure 5-1. Instrument Driver Function Panel Window ..... 5-2
Figure 5-2. Function Panel Controls ..... 5-4
Figure 5-3. Function Panel Window File Menu ..... 5-7
Figure 5-4. Code Menu ..... 5-9
Figure 5-5. Declare Variable Dialog Box ..... 5-10
Figure 5-6. Select UIR Constant Dialog Box ..... 5-12
Figure 5-7. Select Attribute Constant Dialog Box ..... 5-13
Figure 5-8. Select Attribute Value Dialog Box ..... 5-15
Figure 5-9. Select Variable or Expression Dialog Box ..... 5-16
Figure 5-10. View Menu ..... 5-19
Figure 5-11. Options Menu ..... 5-22
Figure 5-12. Help Menu ..... 5-23

Figure 6-1. Variables Window ..... 6-2
Figure 6-2. Watch Window ..... 6-4
Figure 6-3. Add/Edit Watch Expression Dialog Box ..... 6-4
Figure 6-4. File Menu ..... 6-5
Figure 6-5. Edit Menu in the Variables Window ..... 6-6
Figure 6-6. Find Dialog Box in the Variables Window ..... 6-7
Figure 6-7. Find Button Bar ..... 6-8
Figure 6-8. Edit Menu in the Watch Window ..... 6-9
Figure 6-9. View Menu ..... 6-10
Figure 6-10. A Closed Array in the Variables Window ..... 6-10

Figure 6-11. An Expanded Array in the Variables Window ..... 6-11
Figure 6-12. A Parent Structure Pointer in a Chain ..... 6-11
Figure 6-13. A Child Structure Pointer in a Chain ..... 6-12
Figure 6-14. Format Menu ..... 6-13
Figure 6-15. Options Menu ..... 6-14

Figure 7-1. Array Display for a Double-Precision Array ..... 7-2
Figure 7-2. Array Display for a Three-Dimensional Array ..... 7-3
Figure 7-3. Reset Indices Dialog Box for a Three-Dimensional Array ..... 7-3
Figure 7-4. String Display for a String Variable ..... 7-4
Figure 7-5. File Menu ..... 7-5
Figure 7-6. Edit Menu for the Array Display Window ..... 7-6
Figure 7-7. Find Dialog Box in the Array and String Display Windows ..... 7-6
Figure 7-8. Edit Menu for the String Display Window ..... 7-7
Figure 7-9. Format Menu ..... 7-8
Figure 7-10. Format Menu for a Real Array in the Array Display Window ..... 7-8
Figure 7-11. Options Menu ..... 7-9

## Tables

Table 1-1. LabWindows/CVI Startup Options ..... 1-1
Table 1-2. Subdirectories That LabWindows/CVI Requires ..... 1-5

Table 3-1. Platforms Where Utility Functions Require the Low-Level Support Driver ..... 3-26
Table 3-2. VXIplug&play Framework Subdirectories ..... 3-35
Table 3-3. Libraries in the Bin Directory of LabWindows/CVI ..... 3-66

Table 4-1. Regular Expression Characters ..... 4-15
Table 4-2. Keyboard Commands for Implementing Find ..... 4-17
Table 4-3. Keyboard Commands for Implementing Replace ..... 4-18

Table A-1. Keyboard Help ..... A-1

# About This Manual

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

Begin by reading Chapter 1, *Configuring LabWindows/CVI*, and Chapter 2, *LabWindows/CVI Overview*, because subsequent chapters build upon the information in the first two chapters.

## Organization of This Manual

The *LabWindows/CVI User Manual* is organized as follows:

- Chapter 1, *Configuring LabWindows/CVI*, 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.
- Chapter 2, *LabWindows/CVI Overview*, describes the components of LabWindows/CVI, including the LabWindows/CVI environment, and how to create applications with LabWindows/CVI.
- Chapter 3, *Project Window*, describes the LabWindows/CVI Project window, which controls specific tasks related to organizing and executing application programs.
- Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, describes the LabWindows/CVI Source, Interactive Execution, and Standard Input/Output windows. Each of these windows supports specific tasks related to developing and executing programs.
- Chapter 5, *Using Function Panels*, describes how to use LabWindows/CVI function panels to generate code to call functions in any of the LabWindows/CVI libraries.
- Chapter 6, *Variables and Watch Windows*, describes the Variables and Watch Windows. You use these windows to inspect and modify the values of program variables.
- Chapter 7, *Array and String Display Windows*, describes the Array and String Display windows. Use these windows to inspect and modify the contents of a single array or string during a breakpoint.

- Appendix A, *Source Window Keyboard Commands*, can help you quickly identify common Source window keyboard commands that are not in the menus.
- Appendix B, *Customer Communication*, contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation.
- The *Glossary* contains an alphabetical list of terms used in this manual and a description of each.
- The *Index* contains an alphabetical list of key terms and topics used in this manual, including the page where each one can be found.

# Conventions Used in This Manual

This manual uses the following conventions:

| | |
|---|---|
| <> | Angle brackets enclose the name of a key on the keyboard—for example, <Shift>. |
| - | A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Ctrl-Alt-Delete>. |
| **»** | The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options» Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** options from the last dialog box. |
| ☞ | This icon to the left of bold italicized text denotes a note, which alerts you to important information. |
| ⚠ | This icon to the left of bold italicized text denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash. |
| **bold** | Bold text denotes the names of menus, menu items, parameters, or dialog box buttons. |
| ***bold italic*** | Bold italic text denotes an activity objective, note, caution, or warning. |
| *italic* | Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value. |

| | |
|---|---|
| `monospace` | Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, functions, filenames and extensions, and for statements and comments taken from programs. |
| **`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 that you must enter the appropriate words or values in the place of these items. |
| paths | Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files. |

# LabWindows/CVI Documentation Set

## Standard Documentation Set

You should begin by reading *Getting Started with LabWindows/CVI*, which gives you a hands-on introduction to LabWindows/CVI. This manual shows you how to develop applications in LabWindows/CVI.

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

The *LabWindows/CVI Standard Libraries Reference Manual* describes the LabWindows/CVI standard libraries. The *LabWindows/CVI Standard Libraries Reference Manual* assumes that you are familiar with the material presented in *Getting Started with LabWindows/CVI* and the *LabWindows/CVI User Manual*.

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

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

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

The *LabWindows/CVI Advanced Analysis Library Reference Manual* describes a library of advanced analysis functions.

## Related Documentation

The *NI-488.2 Function Reference Manual for DOS/Windows*, the *NI-488.2M Function Reference Manual for Win32*, and the *NI-488.2M Software Reference Manual* describe functions you can use to program National Instruments GPIB interfaces. These manuals are distributed with National Instruments GPIB interface products.

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

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

# Customer Communication

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

# 1

# Configuring LabWindows/CVI

This chapter 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 set the amount of memory for LabWindows/CVI to use. You might not have 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 read 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 Table 1-1.

**Table 1-1.** LabWindows/CVI Startup Options

| 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 in LabWindows/CVI. |
| `-run` | This option automatically invokes the **Run Project** command from the **Run** menu of LabWindows/CVI. |
| `-run_then_exit` | This option automatically invokes the **Run Project** command from the **Run** menu and then automatically invokes the **Exit LabWindows/CVI** command from the **File** menu when the project terminates. This option also suppresses the LabWindows/CVI startup screen and Project window. |

**Table 1-1.** LabWindows/CVI Startup Options (Continued)

| Option | Purpose |
|---|---|
| `-newproject` | LabWindows/CVI starts with an empty Project window. |
| `-p`*`ProcessID`* | This option is available only under Windows 95/NT. LabWindows/CVI attaches to the process that *`ProcessID`* identifies. When the process subsequently loads DLLs, LabWindows/CVI can debug them if you created them in LabWindows/CVI for debugging. You can express *`ProcessID`* as a decimal number or as a hexadecimal number that you precede with `0x`. |

# How to Set the Configuration Options

## Windows 95/NT

Under Windows 95/NT, LabWindows/CVI configuration options are in the Registry under the following key:

`HKEY_LOCAL_MACHINE\Software\National Instruments\CVI`

A configuration string value is associated with each option, as shown in Figure 1-1.

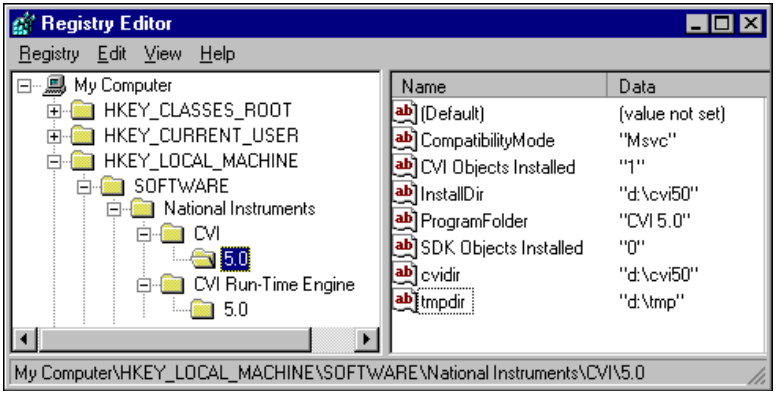


**Figure 1-1.** Registry for Windows 95

You do not have to include an unused configuration string in the Registry.

You must specify an absolute path name, including a drive letter, for configuration strings that take a directory name.

## Windows 3.1

Under Windows 3.1, LabWindows/CVI configuration options are in the `win.ini` file. A configuration string is associated with each option, as shown in the following example.

```
[cvi]
cvidir=
cfgdir=c:\localdir
tmpdir=c:\mytmpdir
CatchProtectionFaults=yes
LoadCVIDebugVxD=yes
```

You do not have to include an unused configuration string in `win.ini`. Leaving the space to the right of the equals sign (=) empty has the same result as not including the configuration string.

You must specify an absolute path name, including a drive letter, for configuration strings that take a directory name.

## UNIX

Under UNIX, LabWindows/CVI configuration options are in the `.Xdefaults` file. The options do not take effect until you restart your X server, or until you use the `xrdb` command to load your `.Xdefaults` file.

A configuration string is associated with each option, as shown in the following example.

```
cvi.cvidir:
cvi.cfgdir:/home/myhomedir/localdir
cvi.tmpdir:/home/myhomedir/mytmpdir
```

You do not have to include an unused configuration string in the `.Xdefaults` file. Leaving the space to the right of the colon (`:`) empty has the same result as not including the configuration string.

You must specify an absolute path name for configuration strings that take a directory name.

# Cross-Platform Option Descriptions

You have the following options when configuring LabWindows/CVI on any platform.

## Directory Options

### cfgdir

`cfgdir` sets the location for the LabWindows/CVI environment settings file, `cvi.ini` under Windows 3.1, and `.cvi.ini` under UNIX, that contains the final settings from the previous session of LabWindows/CVI.

#### Using cfgdir under Windows 3.1

If you do not specify a directory, LabWindows/CVI assumes that the LabWindows/CVI directory contains `cvi.ini`.

If you run LabWindows/CVI across a network, you must set `cfgdir` to one of your local directories.

#### Using cfgdir under UNIX

If you do not specify a directory, LabWindows/CVI assumes that your home directory contains the environment settings file `cvi.ini`.

#### Environment Settings Stored in Registry under Windows 95/NT

The LabWindows/CVI environment does not use `cvi.ini` under Windows 95/NT. Instead, it stores its settings in the Registry under the following key:

```
HKEY_CURRENT_USER\Software\National Instruments\CVI
```

### cvidir

You have to set the `cvidir` option only if the subdirectories that LabWindows/CVI requires, shown in Table 1-2, are not in the directory that contains the LabWindows/CVI executable. The `cvidir` option specifies the directory that contains the subdirectories.

**Table 1-2.** Subdirectories That LabWindows/CVI Requires

| Name of Directory | Contents |
|---|---|
| bin | Resource files (cvi.rsc, cvimsgs.txt), National Instruments Function Panels (.lfp files), National Instruments Libraries (.obj and .lib in Windows, .o and .a under UNIX). |
| font | Font description files. |
| include | C header files for National Instruments libraries. |
| sdk | Windows 95/NT only. Windows SDK. |
| hyperhelp | UNIX only. HyperHelp viewer and associated files. |

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 Table 1-2.

## resdir [obsolete]

The resdir option, which LabWindows/CVI for Windows 3.1 formerly used, is obsolete. If you want to place the LabWindows/CVI executable in a directory other than the installation directory, use the cvidir option.

## tmpdir

tmpdir sets the location for temporary files.

### Using tmpdir in Windows

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.

### Using tmpdir under UNIX

If you do not specify a directory, LabWindows/CVI uses the value of the environment variable TMPDIR. If LabWindows/CVI does not find TMPDIR, or it is invalid, LabWindows/CVI uses /tmp as the location for temporary files.

### Date and Time Options: DSTRules

This option allows you to specify the portions of the year in which daylight savings time is in effect in your area. This affects ANSI C Library functions such as `mktime` and `localtime`. Refer to the *Time and Date Functions* discussion in Chapter 1, *ANSI C Library*, of the *LabWindows/CVI Standard Libraries Reference Manual.*

## Windows Option Descriptions

You have the following additional options when configuring LabWindows/CVI under Windows.

### Timer Options: useDefaultTimer

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 3.1 and Windows 95, and 10 ms under Windows NT.

If you set `useDefaultTimer` to `False` under Windows 3.1 and Windows 95, LabWindows/CVI uses the Windows multimedia library timer. The multimedia timer provides a resolution of 1 ms.

If you set `useDefaultTimer` to `False` under Windows NT, 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 timer, which provides a resolution of 1 ms.

The default value for `useDefaultTimer` is `False`.

### Font Options

Under Windows, LabWindows/CVI provides configuration options to set the fonts that LabWindows/CVI uses in dialog boxes.

#### DialogFontName

`DialogFontName` specifies the font LabWindows/CVI uses in dialog boxes and the built-in pop-up panels, as in the following example: `DialogFontName=Courier`.

### DialogFontSize

`DialogFontSize` specifies the font size LabWindows/CVI uses in dialog boxes and the built-in pop-up panels, as in the following example: `DialogFontSize=30`.

### DialogFontBold

`DialogFontBold` specifies whether the font LabWindows/CVI uses in dialog boxes and the built-in pop-up panels is bold, as in the following example: `DialogFontBold=Yes`.

## Debug Options (Windows 3.1 Only)

### DisplayCVIDebugVxDMissingMessage

When you start LabWindows/CVI, you see the `missing cvidebug.386` message if `LoadCVIDebugVxD` is set to `yes` or is not in the [`cvi`] section of `win.ini`, and the following line is not in the [`386Enh`] section of `system.ini`:

```
device=CVIDebug.386
```

However, if the following line is in the [`cvi`] section of `win.ini`, LabWindows/CVI does not display the message.

```
DisplayCVIDebugVxDMissingMessage=no
```

When you do not specify a value in the `win.ini` file, the default value is `yes`.

### CatchProtectionFaults

`CatchProtectionFaults` determines whether LabWindows/CVI traps general protection faults while LabWindows/CVI is running. Set this value to `no` if you do not want LabWindows/CVI to catch protection faults. If you set this value `no`, using the Variables window might cause LabWindows/CVI to terminate abnormally.

When you do not specify a value in the `win.ini` file, the default is `yes`.

### LoadCVIDebugVxD

`LoadCVIDebugVxD` determines whether LabWindows/CVI uses `cvidebug.386`. If you set this entry to `no`, LabWindows/CVI does not catch general protection faults and the <Ctrl-Alt-SysRq> key combination does not interrupt a running program. If this entry is not present, it defaults to `yes`.

# UNIX Option Descriptions

You have the following additional options when configuring LabWindows/CVI under UNIX.

## Font Options

Under UNIX, LabWindows/CVI provides configuration options to set the standard fonts LabWindows/CVI uses in dialog boxes.

### dialogFont

This option specifies the font NIDialog, which LabWindows/CVI uses in dialog boxes and built-in pop-up panels. The default is `cvi.dialogFont: adobe-helvetica`.

### editorFont

This option specifies the font NIEditor, which LabWindows/CVI uses in the Source window. This font must have a fixed width. The default is `cvi.editorFont: adobe-courier`.

### menuFont

This option, formerly called systemFont, specifies the font NIMenu, which LabWindows/CVI uses in menus. The default is `cvi.menuFont: adobe-helvetica`.

### appFont

This option specifies the font NIApp, which LabWindows/CVI uses in the Project window. The default is `cvi.appFont: adobe-helvetica`.

### messageBoxFont

This option specifies the font NIMessageBox, which LabWindows/CVI uses in message boxes. The default is `cvi.messageBoxFont: adobe-helvetica.`

## Display, Mouse, and Keyboard Options

### activate

When you set this option to `True`, as in `cvi.activate: True`, LabWindows/CVI moves the input focus to a window when you bring it to the front.

The default value is `False`. Use this option only when `ClickToType` is in effect.

## useDefaultColors

If you set this option to True, LabWindows/CVI bases its color palette on the colors that the default color map contains at startup time instead of using its own set of colors. The default value for `useDefaultColors` is `False`.

You might want to use this option if color-intensive applications typically run when you start LabWindows/CVI. In such cases, the default color map might be full, and LabWindows/CVI must use a private color map. Under such circumstances, the system cannot honor the color maps of LabWindows/CVI and the other application simultaneously, so it resorts to using the color map of whichever application owns the active window. When you set this option to `True`, LabWindows/CVI always uses the default color map. As a result, certain colors in the LabWindows/CVI color palette might appear differently depending upon which other applications run before you launch LabWindows/CVI.

## useMetaKey

If you set this option to `True`, LabWindows/CVI uses <Meta> as the command accelerator key. You combine the command accelerator key with other keys to produce shortcut keys. Thus, for example, if you set the `useMetaKey` option to `True`, the shortcut key for the **File Save** command is <Meta-S>. If, on the other hand, you set this option to `False`, LabWindows/CVI uses <Control> as the command accelerator, and <Meta> works the same as <Alt>, which is the key that selects menus and menu items based on the underlined letter in the name. The default value for `useMetaKey` is `False`.

In the LabWindows/CVI development environment, you can override the `useMetaKey` option setting. Use the **Environment Options** command of the Project window to bring up the Keyboard Options dialog box.

## warpMouseOverDialogBoxes

If you set this option to `True`, LabWindows/CVI automatically moves the mouse cursor over dialog boxes when it initially displays them. The default value for `warpMouseOverDialogBoxes` is `True`.

In the LabWindows/CVI development environment, you can override the `warpMouseOverDialogBoxes` option setting. Use the **Environment Options** command of the Project window.

# 2

# LabWindows/CVI Overview

This chapter describes the components of LabWindows/CVI, including the LabWindows/CVI environment, and how to create applications with LabWindows/CVI.

## 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 has the following components.

- Standard libraries and interactive function panels for:
  - GPIB
  - RS-232
  - VISA
  - Data acquisition (distributed with National Instruments PC-based data acquisition boards)
  - Data analysis
  - Transport Control Protocol (TCP)
  - Using X Client Properties for interprocess communication
  - Windows Dynamic Data Exchange (DDE) communication
  - File I/O
  - Data formatting
  - ANSI C
- 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 Automation servers
- A wizard and library for creating IVI instrument drivers. These drivers are highly structured VXI*plug&play*-compatible instrument drivers that use an attribute model to enable advanced features such as state-caching, simulation, and interchangeability

- A set of instrument drivers that contains high-level functions and interactive function panels for controlling specific instruments
- A development environment with windows to manage projects and source code with complete editing, debugging, and user-protection features

Two additional libraries, the VXI Library and the Advanced Analysis Library, are available for LabWindows/CVI. These libraries are optional packages that you can order from National Instruments.

## Standard Libraries

The standard LabWindows/CVI libraries are as follows:

- User Interface Library
- Analysis Library
- GPIB-488/488.2 Library
- RS-232 Library
- Easy I/O for DAQ Library
- VISA Library
- IVI Library
- TCP Library
- X Property Library (UNIX Only)
- DDE Library (Windows Only)
- ActiveX Automation Library
- Formatting and I/O Library
- Utility Library
- ANSI C Library

The functions that make up these libraries can be executed in the LabWindows/CVI environment. You can find descriptions of these library functions in the following manuals:

- *LabWindows/CVI Standard Libraries Reference Manual*
- *LabWindows/CVI User Interface Reference Manual*
- *LabWindows/CVI Instrument Driver Developers Guide*
- *NI-488.2 Software Reference Manual* or *NI-488.2M Software Reference Manual*
- *NI-VISA User Manual* (available upon request)
- *NI-VISA Programmer Reference Manual* (available upon request)

## User Interface Library

You can use the User Interface Library in conjunction with the User Interface Editor in the LabWindows/CVI environment. In the User Interface Editor, you can create command bars, pull-down menus, dialog boxes, controls, graphs, and strip charts. Then you save these objects to a User Interface Resource (`.uir`) file. The functions in the User Interface Library allow you to load these objects from the `.uir` file, display them, receive user input from them, and display program data and results in them. The User Interface Library also has functions for programmatic creation of a Graphical User Interface. The *LabWindows/CVI User Interface Reference Manual* describes the User Interface Library and Editor in detail.

## Data Acquisition Library and Easy I/O for DAQ Library

The Data Acquisition Library, which comes with National Instruments data acquisition boards, contains high-level functions for controlling National Instruments plug-in data acquisition boards. The *NI-DAQ Function Reference Manual for PC Compatibles*, distributed with National Instruments data acquisition boards, describes the library functions.

The Easy I/O for DAQ Library contains functions that make writing simple DAQ programs easier than if you use the Data Acquisition Library. Although the function panels for the Easy I/O for DAQ Library come with LabWindows/CVI, the library requires the NI-DAQ DLL, which comes with your National Instruments data acquisition board. Chapter 10 of the *LabWindows/CVI Standard Libraries Reference Manual* describes the library functions.

## VISA Library and IVI Library

The VISA (Virtual Instrument Software Architecture) Library gives VXI and GPIB software developers, particularly instrument driver developers, a single interface library for controlling VXI, GPIB, RS-232, and other types of instruments. The *NI-VISA Programmer Reference Manual*, available upon request, describes the functions.

The IVI (Intelligent Virtual Instruments) Library gives developers a structured framework for creating VXI*plug&play* instrument drivers with advanced features such as state caching, simulation, and compatibility with generic instrument classes. The Create IVI Instrument Driver wizard supplements the library, automatically creating the skeleton of an IVI driver for you that includes source code and function panels. The *LabWindows/CVI Instrument Driver Developers Guide* contains the IVI Library function reference and instructions on how to create IVI drivers.

## Instrument Library

The Instrument Library is a set of instrument drivers, that each contain high-level C functions for controlling a specific GPIB, RS-232, or VXI instrument. The high-level functions encapsulate the low-level steps necessary to control the instrument and read data. You can use

instrument drivers in the environment in the same way you use the other LabWindows/CVI libraries.

## LabWindows/CVI Environment

The LabWindows/CVI environment makes it easy for you 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. A special feature called a *function panel* makes 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 online help information 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.

- The *Project* window, which appears when you start LabWindows/CVI. You use this window to open, edit, build, run, and save application project (`.prj`) files. A project file is a list of files your application uses. Chapter 3 of this manual, *Project Window*, describes the Project window in detail.
- *Source* windows, which you use to create, edit, run, debug, and save source code. This window includes an optional toolbar to give you quick access to commands you use frequently. Chapter 4 of this manual, *Source, Interactive Execution, and Standard Input/Output Windows*, describes Source windows in detail.
- *Function Panel* windows, which you use to interactively execute library functions and insert code into Source windows. This window includes an optional toolbar to give you quick access to commands you use frequently. Chapter 5 of this manual, *Using Function Panels*, describes Function Panel windows.
- The *Interactive Execution* window, which you use to execute selected portions of code. You do not have 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. Chapter 4 of this manual, *Source, Interactive Execution, and Standard Input/Output Windows*, describes this window in detail.
- *Variables*, *Array Display*, *String Display*, and *Watch* windows, which you use for debugging programs. Chapter 6, *Variables and Watch Windows*, and Chapter 7, *Array and String Display Windows*, of this manual, describe these windows.

- *User Interface Editor* windows, which you use to build graphics-mode command bars, pull-down menus, dialog boxes, controls, graphs, and strip charts and save them to User Interface Resource (`.uir`) files. The *LabWindows/CVI User Interface Reference Manual* describes the User Interface Editor window.
- *Function Tree Editor* windows, which you use to build the tree structure of function panel files. The *LabWindows/CVI Instrument Driver Developers Guide* describes Function Tree Editor windows.
- *Function Panel Editor* windows, which you use to build function panels. This window includes an optional toolbar to give you quick access to commands you use frequently. The *LabWindows/CVI Instrument Driver Developers Guide* describes Function Panel Editor windows.
- *Function Tree Help Editor* and *Function Panel Help Editor* windows, which you use to add online help to function panels. The *LabWindows/CVI Instrument Driver Developers Guide* describes these windows.
- The *Standard Input/Output* window, which you use for printing text messages and receiving user input from the keyboard. Chapter 4 of this manual, *Source, Interactive Execution, and Standard Input/Output Windows*, describes this window in detail.

You develop applications in the LabWindows/CVI environment using the ANSI C programming language. For information on the LabWindows/CVI compiler/linker and how to use the LabWindows/CVI libraries with other compilers and linkers, refer to the *LabWindows/CVI Programmer Reference Manual*.

# How to Create Applications with LabWindows/CVI

Use LabWindows/CVI as a text editor in which to enter your entire program. You can greatly simplify application development by using function panels to execute LabWindows/CVI functions and automatically insert the code into your program. Function panels contain complete online help. Refer to Chapter 5, *Using Function Panels*, for more details.

The Project window contains all the component files of your application. The simplest case is one source file, as shown in Figure 2-1.

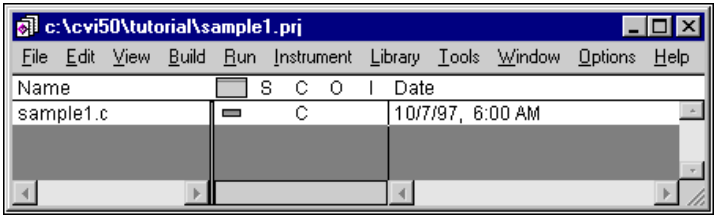


**Figure 2-1.** Project Window

A typical project, however, contains multiple code modules and a User Interface Resource file. You can list code modules as source files or compiled files. LabWindows/CVI recompiles source files each time you open the project, thereby increasing initial project startup time. However, 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. Refer to the *LabWindows/CVI Programmer Reference Manual* for more information on compatible external compilers. LabWindows/CVI does not recompile compiled files each time you open the project, reducing initial project startup time. Also, 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 list to be compiled into an object module. In this way, LabWindows/CVI automatically compiles the file for you, but you do not have to recompile the file each time you open the project.

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 list in your project.

## Creating a User Interface

You can create user interface objects (panels, controls, menus) using the User Interface Editor window 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.

The LabWindows/CVI CodeBuilder automatically generates complete C code that compiles and runs based on a user interface (`.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 in 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. For more information, refer to the *CodeBuilder Overview* section in Chapter 2 of the *LabWindows/CVI User Interface Reference Manual*.

## Creating Standalone Programs and DLLs

With the LabWindows/CVI Run-time Engine, you can create standalone executables, dynamic link libraries, and static libraries. Chapter 7, *Creating and Distributing Standalone Executables and DLLs*, in the *LabWindows/CVI Programmer Reference Manual* describes the engine.

# 3

# Project Window

This chapter describes the LabWindows/CVI Project window, which controls specific tasks related to organizing and executing application programs.

## Project Window Overview

Use the Project window to open, edit, build, run, and save application project (`.prj`) files. A project file is a list of files your application uses. Certain files must be in the list, while others are optional. If you had a project loaded the last time you used LabWindows/CVI, that project appears in the Project window when you start LabWindows/CVI again.

Unless you use the following files as instrument driver program files or load them dynamically using `LoadExternalModule`, you must put them in your project file list.

- Source files your application program uses, ending with the `.c` extension
- Object files your application program uses, ending with `.obj` under Windows or `.o` under UNIX
- Library files your application program uses, ending with `.lib` under Windows or `.a` under UNIX. DLL import libraries for Windows 95/NT are in this category
- Dynamic-link library files your application program uses, under Windows 3.1 only, ending with the `.dll` extension

The following files are optional in your project file list.

- Header files (`.h`) your application program uses. Listing `.h` files makes it easy to open them for viewing or editing, and ensures that the compiler can find them.
- User interface resource files (`.uir`) your application program uses. Listing `.uir` files makes it easy to open them for viewing or editing, and ensures that LabWindows/CVI can find them.
- Instrument driver function panel files (`.fp`). 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. Refer to the *Using Instrument Drivers* and *Instrument Menu* sections in this chapter for information about instrument driver program files.

Figure 3-1 shows a sample Project window.

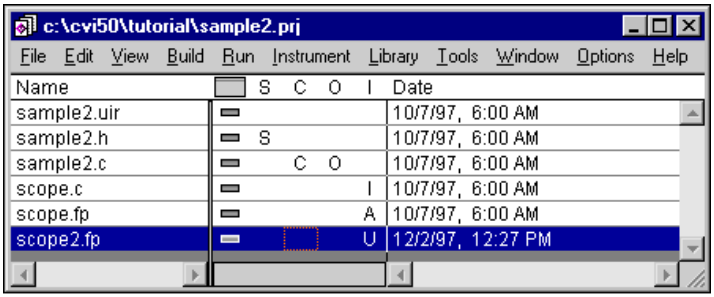


**Figure 3-1.** Project Window

You can open `.c`, `.h`, and `.uir` files in the project list by double-clicking directly on the file name. Double-clicking on a `.fp` filename opens the Function Panel Selection dialog box for the instrument driver.

You can use five icons in the Project window, as described below.

The file is currently closed. Double-click on this icon to open the file. If the file is a `.fp` file, double clicking opens the Function Tree Editor.

The file is currently open. Double-click on this icon to close the file. If the file is a `.fp` file, double clicking hides the Function Tree Editor window, but does not unload the `.fp` file.

The file has been modified since you last saved it. Double-click on this icon to save the file.

You have not compiled the file since you loaded the project, you have modified the file since you last compiled it, or you manually marked it for compilation. Double-click on this icon to compile the file.

This icon applies only to source (`.c`) files and indicates that you enabled the Compile into Object option. If this option is enabled when you compile the source file, LabWindows/CVI creates an object (`.obj` or `.o`) file on disk that contains non-debuggable code rather than generating debuggable code in memory. When you load a project that contains source files with the Compile into Object option enabled, those source files do not have to be compiled unless you have made changes to them on disk since you last saved the project. Double click on this icon to toggle the option.



The file is associated with a loaded instrument driver.

A This icon indicates that the `.fp` file is loaded into the **Instrument** menu and signifies the following: Attached to (or Associated with) a program file.

U This icon indicates that the `.fp` file is loaded into the **Instrument** menu and signifies the following: Unattached to any program file. When you double-click on this icon, LabWindows/CVI tries to attach a program file.

If no icon appears in the `I` column next to a `.fp` file, the `.fp` file is not loaded into memory. When you double-click on this icon, LabWindows/CVI tries to load the `.fp` file into memory and attach the instrument driver program file.

# File Menu

This section contains a detailed description of the Project window **File** menu, as shown in Figure 3-2.

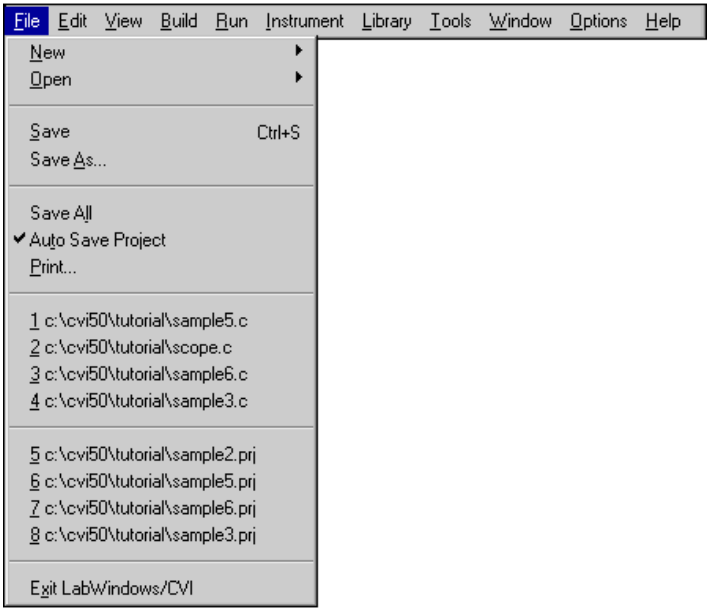


**Figure 3-2.** File Menu

## New

The **New** command has a submenu, as shown in Figure 3-3.

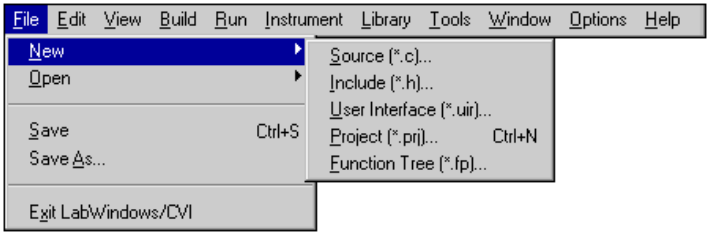


**Figure 3-3.** New Command Submenu

Use the **New** command to open various types of 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. Refer to the *LabWindows/CVI User Interface Reference Manual* for more information about User Interface Editor windows.

If you choose **Project**, a dialog box appears with a message that asks if you want to unload the current project. You can work with only one project at a time. If you select **Yes**, a new Project window appears. You are prompted to save any modified files in the old project. You are also prompted to save project options. The *Options Menu* section later in this chapter describes these options.

If you choose **Function Tree**, a new Function Tree Editor window appears in which you can create a new `.fp` file. Refer to the *LabWindows/CVI Instrument Driver Developers Guide* for details about Function Tree Editor windows.

## Open

The **Open** command has a submenu, as shown in Figure 3-4.

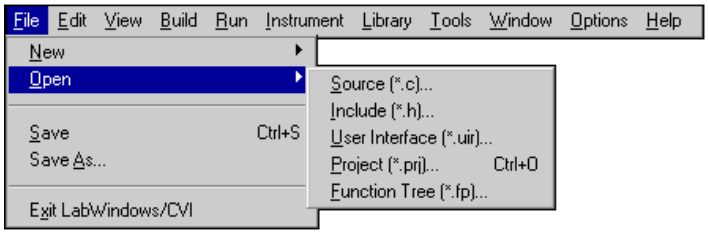


**Figure 3-4.** Open Command

Use the **Open** command to open various types of user specified files. When you select **Open**, a dialog box appears, prompting you for a filename to load into a new window. One feature of this dialog box is the Directories ring, which is in the upper-right corner of the dialog box under Windows 3.1, NT, and UNIX, and at the top of the dialog box under Windows 95. When activated, this 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**, a User Interface Editor window appears with your specified `.uir` file.

If you choose **Project**, a Project window appears with your specified `.prj` file. LabWindows/CVI prompts you to save any modified files in the old project.

If you choose **Function Tree**, a new Function Tree Editor window appears with your specified `.fp` file. You cannot use this command to load instrument modules. You load instrument modules from the **Instrument** menu.

## Save

Use the **Save** command to write the project (`.prj`) file to disk. If you want to append a different extension, type it in after the filename. If you do not want to append any extension, enter a period after the filename.

## Save As. . .

Use the **Save As** command to write the contents of the Project window to disk using a new name you specify and change the name on the Project window title bar to your new name. If you want an extension other than `.prj`, type it in after the filename. If you do not want to append any extension, enter a period after the filename.

## Save All

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

## Auto Save Project

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

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

Notice that if you disable the **Auto Save Project** command, LabWindows/CVI does not save the project file when you start running a program, even if you set the Save Changes before Running option in the Run Options dialog box to Always or Ask.

## Print

The **Print** command opens a list of all the printable files in the project. You can select the files you want to print.

## 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 project files
- A list of the four most recently closed project files

## Exit LabWindows/CVI

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

# Edit Menu

This section contains a detailed description of the Project window **Edit** menu, as shown in Figure 3-5.

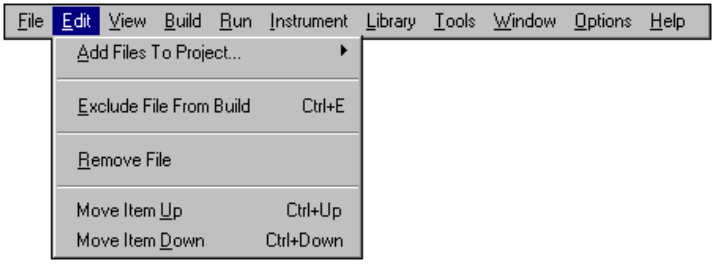


**Figure 3-5.** Edit Menu

## Add Files to Project. . .

The **Add Files to Project** command has a submenu, as shown in Figure 3-6.

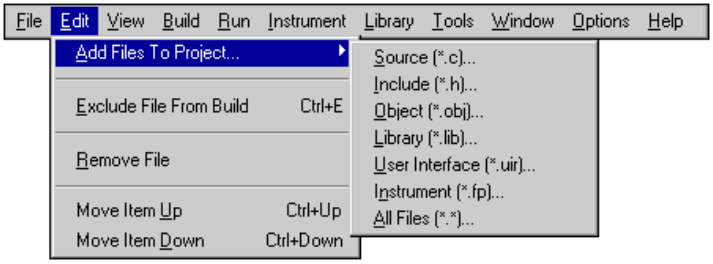


**Figure 3-6.** Add Files to Project Command Submenu

Use the **Add Files to Project** item to add any type of file to the project list. Choose any one of the file types listed in the tiered menu to invoke a dialog box for selecting that type of file.

Use the **Source**, **Object**, **Library**, and **DLL** items to add code modules to your project. Refer to Chapter 2, *Using Loadable Compiled Modules*, in the *LabWindows/CVI Programmer Reference Manual* for information about using object, library, and DLL files in LabWindows/CVI.

Use the **Include** item to add header files to your project. It is a good idea to list header files in your project because this makes access to you header files much easier.

Under Windows 3.1, use the **DLL Path** item to add a DLL path (`.pth`) file to your project. You must use DLL path files when you want to load a DLL using the standard Windows DLL search method. Refer to Chapter 4, *Windows 3.1 Compiler/Linker Issues*, in the *LabWindows/CVI Programmer Reference Manual* for more details.

Under Windows 95/NT, an import library (`.lib`) file must accompany each DLL. If you want to use a DLL in your project, you must list the import library rather than the DLL. You cannot add DLL and DLL path (`.pth`) files to the project under Windows 95/NT. If you load a project that was created in Windows 3.1 and contains `.dll` or `.pth` files, LabWindows/CVI displays a warning message and excludes the files.

For more detailed information on using DLLs in LabWindows/CVI for Windows 95/NT, refer to the *Loading DLLs in LabWindows/CVI* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, in the *LabWindows/CVI Programmer Reference Manual*.

Use the **User Interface** item to add `.uir` files to your project. You can list `.uir` files in your project to make access to the file easier.

Use the **Function Tree** 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.

## Exclude File from Build/Include File in Build

The **Exclude File from Build** command excludes the highlighted code module file from the build. This command does not apply to `.h`, `.fp`, or `.uir` files. Excluded files appear in a different color in the Build window. LabWindows/CVI does not compile or link them into the project. When you exclude a file, the command toggles to **Include File in Build** so you can include the file in the build again.

## Remove File

Use the **Remove File** command to remove the highlighted file from the project list.

## Move Item Up

Use the **Move Item Up** command to move the highlighted file up one line in the project list. To activate this menu item, select **No Sorting** from the **View** menu.

## Move Item Down

Use the **Move Item Down** command to move the highlighted file down one line in the project list. To activate this menu item, select **No Sorting** from the **View** menu.

# View Menu

This section explains how to use the commands in the Project window **View** menu, as shown in Figure 3-7.

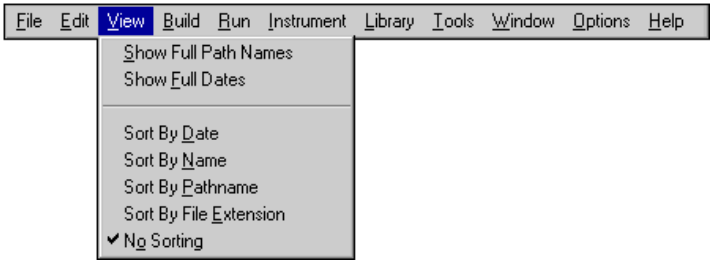


**Figure 3-7.** View Menu

## Show Full Path Names

Use this command to toggle between displaying the project list with full path names and simple base file names.

## Show Full Dates

Use this command to toggle between displaying the project list with short file dates, such as `06/15/93`, and full file dates, such as `Tue, Jun 15, 1993`.

## Sort By Date

If you sort by date, the project list appears in chronological order.

## Sort By Name

If you sort by name, the project list appears in alphabetical order by file name.

## Sort By Pathname

If you sort by pathname, the project list appears in alphabetical order by directory pathname.

## Sort By File Extension

If you sort by file extension, the project list appears in alphabetical order by file extension.

## No Sorting

If you choose **No Sorting**, you can list your project files in any order by using the **Move Item Up** and **Move Item Down** items in the **Edit** menu.

# Build Menu

This section explains how to use the commands in the Project window **Build** menu, as shown in Figure 3-8.

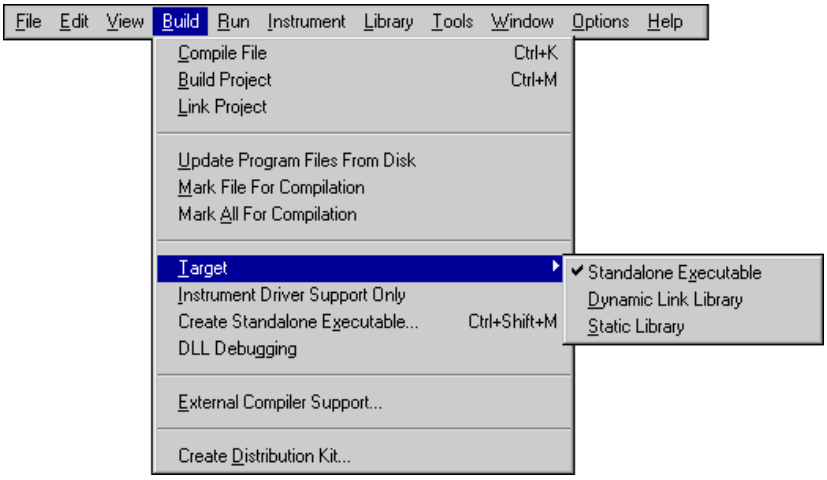


**Figure 3-8.** Build Menu

Use commands in the **Build** menu for compiling files, building and linking projects, marking files for compilation, and creating application files.

## Compile File

You must compile your source code before you execute your project. Use the **Compile File** command to compile the selected source file. If LabWindows/CVI encounters any build errors, the Build Errors window appears with a list of errors.

Refer to the descriptions of **Compiler Options** and **Compiler Defines** in the *Options Menu* section of this chapter for a discussion of compiler options and defines.

## Build Project

The **Build Project** command compiles all source files listed in the project that you or LabWindows/CVI mark for compilation and links the compiled files. Refer to the description of the **Mark File for Compilation** command in this section for information on marking files for compilation.

## Link Project

The **Link Project** command links the compiled files from the project. It does not compile any files.

## Update Program Files from Disk

This command determines if you have modified any `.c`, `.h`, `.obj` (Windows), `.o` (UNIX), `.lib` (Windows), or `.a` (UNIX) files, either in your project or associated with a loaded instrument, outside of LabWindows/CVI. It does so by comparing the recorded dates of the files with the actual dates of the files on disk. If any file in the project, other than `.uir` or `.fp` files, has a different date on disk, the following rules apply:

- If the file is not open, or if the file is open but you have not modified it, LabWindows/CVI reloads the file on disk into the project.
- If the file is open and you have modified it, you can choose to overwrite the disk file with the file window contents, or load the disk file into a new window and close the current window.

## Mark File for Compilation

When LabWindows/CVI marks a source file for compilation, a "**C**" appears next to the filename in the Project 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 with the **Mark File for Compilation** command.

## Mark All for Compilation

Use the **Mark All Files for Compilation** command to force LabWindows/CVI to recompile all source files in the project the next time you build the project.

## Target (Windows 95/NT Only)

The **Target** 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 command that appears below **Target** in the **Build** menu. The **Create** command that appears below **Target** in the **Build** menu changes name depending on the target type you select. The target types that you can select are:

- **Standalone Executable**
- **Dynamic Link Library**
- **Static Library**

When you select anything other than **Standalone Executable**, the **Run Project** command in the **Run** menu dims. If you select **Dynamic Link Library**, you can use the **Select External**

**Process** command in the **Run** menu to specify an external program that uses the DLL. When you do this, the **Run Project** command changes to **Run *xxxx*.exe**, where *xxxx*.exe is the name of the program you specify.

## Instrument Driver Support Only

**Note** ***This command is not available under Windows 3.1.***

If you enable the **Instrument Driver Support Only** command, your project does not link to the entire set of LabWindows/CVI libraries, but to a smaller set of functions. Standalone executables and DLLs you create when the **Instrument Driver Support Only** command is enabled do not use the LabWindows/CVI Run-time Engine DLL, `cvirte.dll`. Instead, they use `instrsup.dll`, which is much smaller. On Solaris 1 and 2, it is called `libinstrsup.so`. On HP-UX, it is called `libinstrsup.sl`.

This command is particularly useful for creating instrument driver DLLs. It allows other applications to use instrument driver DLLs without having to load the large LabWindows/CVI Run-time Engine DLL.

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

- Formatting and I/O Library
- RS-232 Library
- Utility Library (selected functions only—refer to the *Utility Library Functions* discussion later in this section)
- ANSI C

If you use a standalone compiler and want to use `instrsup.dll`, include `cvi\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.

Your project also can link to the following libraries:

- Analysis or Advanced Analysis Library
- GPIB Library
- VXI Library
- VISA Library
- IVI Library
- Easy I/O for DAQ Library
- Data Acquisition Library

If you use a standalone compiler under Windows 95/NT and want to use any of these libraries, refer to the *Using the LabWindows/CVI Libraries in External Compilers* section in Chapter 3, *Windows 95/NT 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 includes `instrsup.dll` in the distribution kit and disables the option to distribute the full LabWindows/CVI Run-time Engine.

### Utility Library Functions

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

```
Beep
DateStr
Delay
SyncWait
Timer
TimeStr
RoundRealToNearestInteger
TruncateRealNumber
InStandaloneExecutable
CVIRTEHasBeenDetached
```

### No Standard Input/Output

`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`.

### Multithread Safe

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

### Generate Makefile (UNIX Only)

You can use this command to generate a UNIX Makefile that corresponds to the currently loaded project and libraries. The Makefile invokes an external compiler to build the application.

## Create Standalone Executable. . .

Use the **Create Standalone Executable** command to create an executable version of the current project. Before creating an executable version of your project, read Chapter 7, *Creating and Distributing Standalone Executable Programs*, in the *LabWindows/CVI*

*Programmer Reference Manual*. When you select the **Create Standalone Executable** command, the Create Standalone Executable dialog box appears, as shown Figure 3-9.

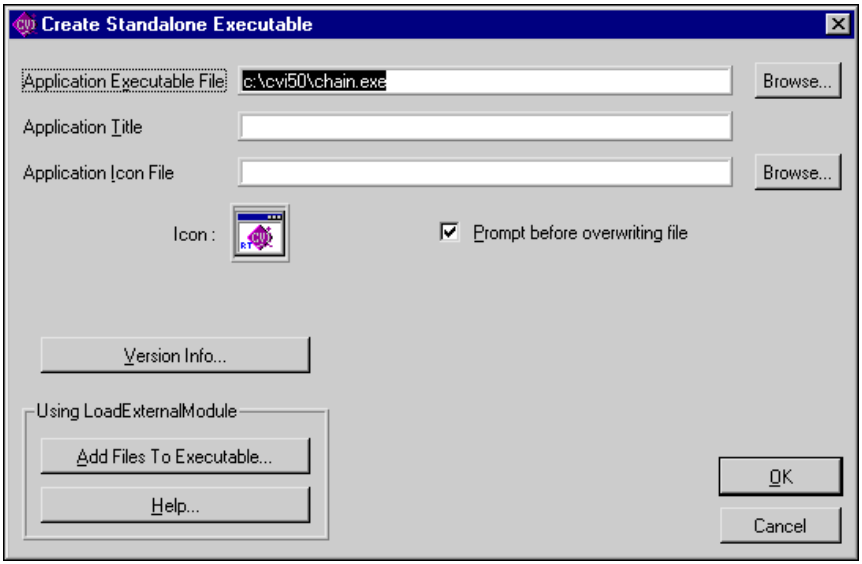


**Figure 3-9.** Create Standalone Executable Dialog Box

- **Application Executable File**—The name of the executable file associated with your program. You can use the **Browse** button to select an existing filename.
- **Application Title**—A descriptive title for your program. This title appears below the icon of the executable program. The operating system registers it when a user starts the application.
- **Application Icon File (Windows only)**—A file that contains a descriptive graphical icon for your program. You can double-click on the icon in the Program Manager to start your executable. You can use the **Browse** button to select an existing icon file.
- **Icon (Windows only)**—The graphical representation of the Application Icon File. You can double-click on this control to browse for an icon file on disk. The sample program, `samples\apps\iconedit.exe`, ships with LabWindows/CVI so that you can create your own icon files.
- **Prompt Before Overwriting Executable File**—A checkbox where you can request LabWindows/CVI to prompt you before it overwrites an Application Executable File that has the same name as the one you are creating.
- **Using LoadExternalModule**
  - **Add Files to Executable**—This button lets you select additional module files you want to link into the Application Executable File. These are modules that your

project files do not directly reference but which are referenced by modules you load at run-time by calling `LoadExternalModule`.

– **Help**—This button describes the use of `LoadExternalModule` in an executable and the **Add Files to Executable** button.

- **OK**—This button accepts the current inputs and attempts to create the Application Executable File.
- **Version Info**—When you click on this button, the Version Info dialog box appears, in which you can enter version information for the executable file. LabWindows/CVI saves the information in the executable in the form of a standard Windows version resource. You can obtain the information from the executable by using the Windows SDK functions `GetFileVersionInfo` and `GetFileVersionInfoSize`.

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

  *n*, *n*, *n*, *n*

  where *n* is a number from 0 to 255
- **Cancel**—This button cancels the operation and removes the dialog box.

## Create Dynamic Link Library (Windows 95/NT Only)

Use the **Create Dynamic Link Library** command to create a dynamic link library (`.dll`) file from the current project. LabWindows/CVI also creates a DLL import library (`.lib`) file. When you select the command, the Create Dynamic Link Library dialog box appears, as shown in Figure 3-10.

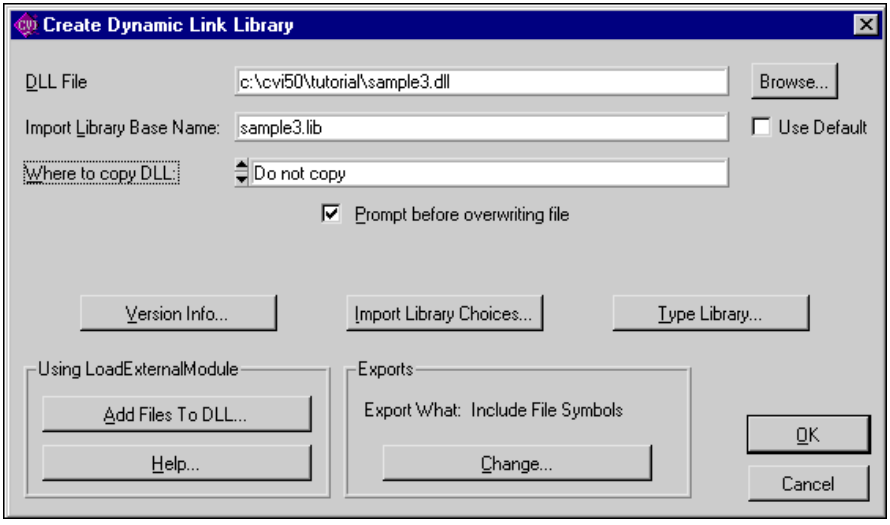


**Figure 3-10.** Create Dynamic Link Library Dialog Box

- **DLL File**—The name of the DLL file to be created. You can use the **Browse** button to select an existing filename or enter a new one.
- **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, where 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* instrument driver DLLs. If you want to enter a different name for the import library, deselect the Use Default checkbox. Enter a name without any directory names.
- **Where to Copy DLL**—A ring control where you can instruct LabWindows/CVI to copy the DLL to a different directory after creating it. Your choices are the following:
  - Do not copy
  - Windows System directory
  - VXI*plug&play* directory (the `bin` directory under the VXI*plug&play* framework directory)
- **Prompt Before Overwriting File**—A checkbox where you can request LabWindows/CVI to prompt you before it overwrites a DLL file that has the same name as the one you are creating.
- **Version Info**—When you click on this button, the Version Info dialog box appears, where you can enter version information for the DLL. LabWindows/CVI saves the version information in the DLL in the form of a standard Windows version resource. You can obtain the information from the DLL by using the Windows SDK functions `GetFileVersionInfo` and `GetFileVersionInfoSize`.

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

  *n*,*n*,*n*,*n*

  where *n* is a number from 0 to 255
- **Import Library Choices**—This button lets you choose whether to create a DLL import library for each of the compatible external compilers, or to create one only for the current compatible compiler. Refer to the *Compatibility with External Compilers* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*. It also lets you choose whether to create the import libraries in the VXI*plug&play* 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`, `borland`, `watcom`, and `symantec`. 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 each compiler, LabWindows/CVI creates the files in the subdirectories `msc`, `bc`, `wc`, and `sc`

under the VXI*plug&play* `lib` directory. If you choose to create an import library for the current compiler only, LabWindows/CVI creates the file in the appropriate subdirectory.

- **Type Library**—This button lets you 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 using the **Generate Windows Help** command in the **Options** menu of the Function Tree Editor window and then using the Windows Help Compiler.

  This feature is useful if you intend for your DLL to be used from Visual Basic. For more information, refer to the *Automatic Inclusion of Type Library Resource for Visual Basic* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

- **Using LoadExternalModule**
  - **Add Files to DLL**—This button lets you select additional module files which 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`.
  - **Help**—This button describes the use of `LoadExternalModule` in a DLL and the **Add Files to DLL** button.
- **Exports**
  - **Export What**—This indicates your current choice of 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**—This button lets you select the method to use for determining which symbols in the DLL to export to the users of the DLL. The choices are the following:
    - **Include File Symbols**—You must name one or more include files that declare symbols defined globally in the DLL. The declared symbols are the ones exported. You can select from a list of include files in the project.
    - **Symbols Marked for Export**—The DLL exports all symbols you define in the DLL with the qualifier `__declspec(dllexport)` or `export`.
- **OK**—This button accepts the current inputs and attempts to create the DLL.
- **Cancel**—This button cancels the operation and removes the dialog box.

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

For more information on creating DLLs, refer to the *Preparing Source Code for Use in a DLL* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

## Create Static Library (Windows 95/NT Only)

Use the **Create Static Library** command to create a static library (`.lib`) file from the current project. When you select this command, the Create Static Library dialog box appears, as shown Figure 3-11.

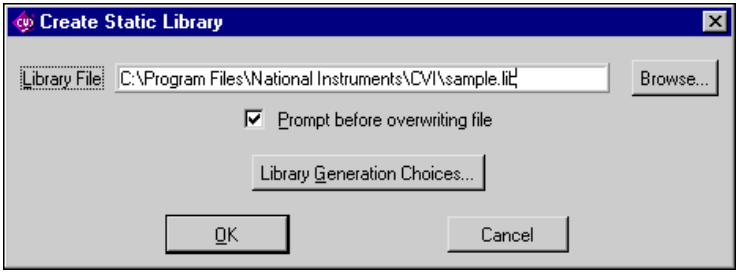


**Figure 3-11.** Create Static Library Dialog Box

- **Library File**—The name of the library file to be created. You can use the **Browse** button to select an existing filename or name a new one.
- **Prompt Before Overwriting File**—A checkbox where you can request LabWindows/CVI to prompt you before it overwrites a library file that has the same name as the one you are creating.
- **Library Generation Choices**—This button lets you choose whether to create a static library for each of the compatible external compilers, or create one for the current compatible compiler only. Refer to the *Compatibility with External Compilers* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*. If you want to create a static library for each compiler, 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`, `borland`, `watcom`, and `symantec`. LabWindows/CVI also creates the library for the current compatible compiler in the parent directory.
- **OK**—This button accepts the current inputs and attempts to create the library file.
- **Cancel**—This button cancels the operation and removes the dialog box.

☞ **Note** ***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 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.***

## DLL Debugging (Windows 95/NT Only)

This command applies only when you set the **Target** item in the **Build** menu to **Dynamic Link Library**. If the **DLL Debugging** command is enabled when you create a DLL, LabWindows/CVI includes debug code in your DLL and generates an extra file that contains a symbol table and source position information necessary for debugging. The extra file has the same pathname as the DLL except that its extension is `.cdd`.

In the LabWindows/CVI development environment, you can debug only DLLs you create in LabWindows/CVI with the **DLL Debugging** command enabled. Other development environments cannot debug DLLs you create in LabWindows/CVI.

When debugging a DLL in LabWindows/CVI, you can use all the standard debugging features, but the run-time checking features are not available. You can thus use single-stepping, conditional breakpoints, watch expressions, and the Variables window; but there is no interactive reporting of library errors and no protection against overwriting arrays or misuse of pointers. When you create a DLL with the **DLL Debugging** command disabled, LabWindows/CVI forces the debugging level to None. When you create a DLL with the **DLL Debugging** command enabled, LabWindows forces the debugging level to No Run-Time Checking. You can access the debugging level from the **Run Options** command in the **Options** menu of the Project window.

## Location of Files Required for Debugging DLLs

To debug a DLL in LabWindows/CVI, the `.cdd` file and the source files for the DLL must be available. LabWindows/CVI looks for the `.cdd` file in the following locations and order:

1. The directory in which you created the DLL
2. The directory from which LabWindows/CVI loaded the DLL
3. The directory of the current project target, if the current project target is the DLL
4. The Windows directory
5. The Windows System directory

If it cannot find the `.cdd` file in any of these locations, a dialog box prompts you to browse for it. After you enter the location of the `.cdd` file, LabWindows/CVI stores the location in the Windows 95/NT registry.

The `.cdd` file contains the locations of the source files at the time you created the DLL. It also contains the LabWindows/CVI installation directory and VXI*plug&play* framework

directory. When LabWindows/CVI has to display a DLL source file, it looks for the file in the following places and order:

1. The project list, if the current project target is the DLL you are debugging
2. The source file directory that LabWindows/CVI stored in the `.cdd` file
3. If you have moved the `.cdd` file, the directory that is in the same relative position to the current `.cdd` location as the stored source file directory was in relation to the original `.cdd` file location
4. If the source file was originally under the LabWindows/CVI directory and the LabWindows/CVI directory has changed, the same relative position to the new LabWindows/CVI directory
5. If the source file was originally under the VXI*plug&play* framework directory and the VXI*plug&play* framework directory has changed, the same relative position to the new VXI*plug&play* framework directory

If LabWindows/CVI cannot find a DLL source file, it reports an error. Make sure that your files are in one of the preceding locations.

In summary, when you move a DLL to another machine and you want to debug it there, you must also copy the `.cdd` file and the source files. It is best to keep the `.cdd` file and source files in the same relative location to each other. You do not have to keep the `.cdd` file in the same directory as the DLL. LabWindows/CVI prompts you for the `.cdd` file location and keeps track of it thereafter.

## Different Ways to Debug DLLs

You can debug a DLL two ways. In one approach, you run a LabWindows/CVI project that calls the DLL. The DLL project is not open in LabWindows/CVI. In the other approach, you open the DLL project and run an external process that uses the DLL.

### Running a Program in LabWindows/CVI

To debug a DLL that another LabWindows/CVI project uses, you must run the project with the debugging level set to a value other than None. When LabWindows/CVI loads the DLL, it loads the corresponding debug information from the `.cdd` file. LabWindows/CVI honors breakpoints you set in DLL source files. LabWindows/CVI saves in the project any breakpoints you set in any source file, regardless of whether the source file is in the project.

Also, you can set watch expressions for a debuggable DLL. For each watch expression, you must choose whether it applies to a project or a DLL. If it applies to a DLL, you must enter the name of the DLL. LabWindows/CVI stores this information in the project. For more information, refer to Chapter 6, *Variables and Watch Windows*, in this manual.

You can debug multiple DLLs called from the same project. LabWindows/CVI handles each DLL in the same manner.

### Running an External Process

To debug a DLL that an external process uses, load the DLL project into LabWindows/CVI. Make sure you set the debugging level to a value other than None. Execute the **Select External Process** command from the **Run** menu of the Project window. A dialog box appears that allows you to enter the pathname of an external process and command line arguments. LabWindows/CVI stores this information in the project.

After you have specified the pathname of an external process, the **Run Project** item in the **Run** menu changes to **Run *xxx*.exe**, where *xxx*.exe is the filename of the external process.

Execute the **Run *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.

If the external process loads other debuggable DLLs, you can debug them even though a project for a different DLL is open. LabWindows/CVI handles the other DLLs as the *Running a Program in LabWindows/CVI* discussion earlier in this section describes.

☞ **Note** ***The command-line arguments that you set in the Select External Process dialog box are the same command line arguments that you set using the* Command Line *item in the* Options *menu of the Project window.***

#### Multithreaded Executables

LabWindows/CVI can debug only one thread. If a multithreaded external process calls functions in debuggable DLLs in more than one thread, LabWindows/CVI can debug only the thread that first loaded a debuggable DLL. LabWindows/CVI does not honor breakpoints and watch expressions in the other threads.

If the external process unloads all the debuggable DLLs and then loads another, LabWindows/CVI can debug only in the thread that loaded the new DLL.

## External Compiler Support (Windows 95/NT Only)

Use the **External Compiler Support** command to help you build your executable or DLL in one of the four compatible external compilers. For detailed information on this topic, refer to the *Creating Executables and DLLs in External Compilers for Use with the LabWindows/CVI Libraries* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

When you execute the command, the External Compiler Support dialog box appears, as shown in Figure 3-12.

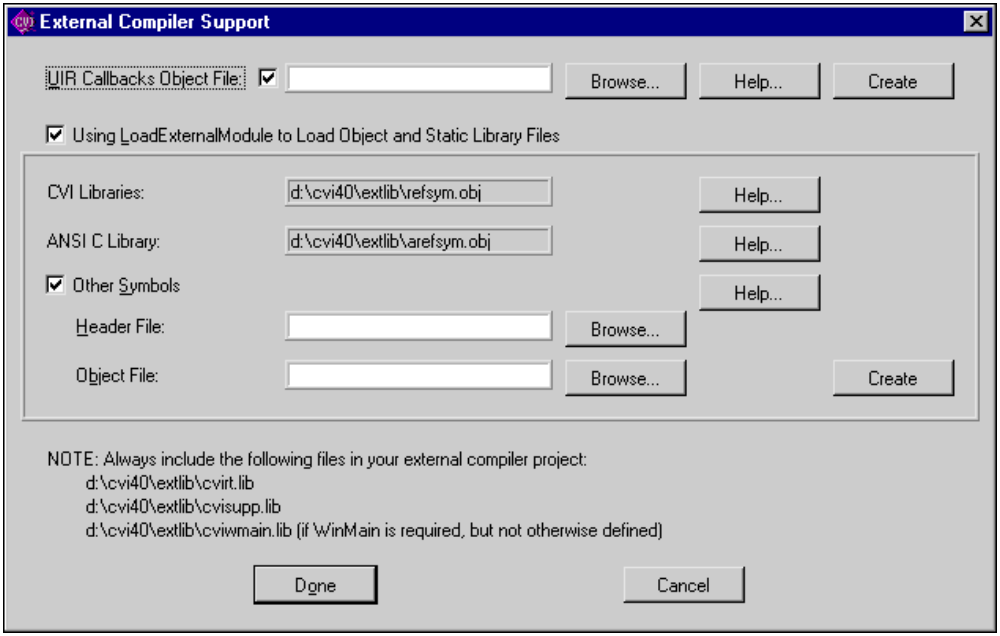


**Figure 3-12.** External Compiler Support Dialog Box

- **UIR Callbacks Object File**—This option creates an object file for you to link into your executable or DLL. The object file contains a list of the callback functions you specify in the User Interface Resource (`.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), checkmark the checkbox, enter the name of the object file to create, and click on the **Create** button. In the future, whenever you save modifications to any of the `.uir` files in the project, LabWindows/CVI automatically updates the object 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 object 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**—This option 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**—This display provides information `LoadExternalModule` requires when your run-time modules reference symbols in any of the following LabWindows/CVI libraries:
  - User Interface
  - RS-232
  - DDE
  - TCP
  - Formatting and I/O
  - Utility

  If you use one of these libraries, include in your external compiler project the object file displayed in this indicator.
- **ANSI C Library**—This display provides information `LoadExternalModule` requires when your run-time modules reference symbols in the ANSI C library. Include in your external compiler project the object file displayed in this indicator.
- **Other Symbols**—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. This option creates an object file for you to link into your executable (or DLL).
  - **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 on the **Create** button 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 for you to include in your external compiler project. The files are the following:

```
cvi\extlib\cvirt.lib
cvi\extlib\cvisupp.lib
cvi\extlib\cviwmain.lib
```

Use `cviwmain.lib` only when the external compiler requires you to define `WinMain`, you do not define it in your project, and 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.

## Create Distribution Kit (Windows Only)

The **Create Distribution Kit** command is available only in LabWindows/CVI for Windows. For information on distributing executables under UNIX, Refer to the *Distributing Standalone Executables Under UNIX* section in Chapter 5, *UNIX Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

Use the **Create Distribution Kit** command to make a set of disks 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 and files that you load using `LoadExternalModule`.

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** features of the **Create Distribution Kit** command. Refer to the *Rules for Loading Files Using LoadExternalModule* section in Chapter 7, *Creating and Distributing Standalone Executables and DLLs*, of the *LabWindows/CVI Programmer Reference Manual*.

When you select the **Create Distribution Kit** command, the Create Distribution Kit dialog box appears, as shown in Figure 3-13.

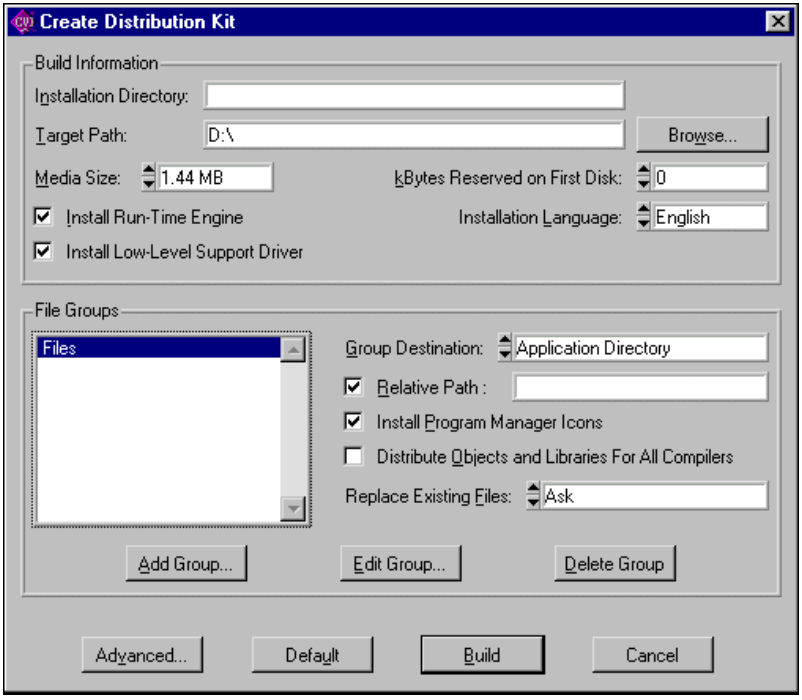


**Figure 3-13.** Create Distribution Kit Dialog Box

## Build Information Section

- **Installation Directory**—The default directory that appears in the installation of the user.
- **Target Path**—The path into which you want to build your distribution kit.

☞ **Note** ***When you define a target path to a floppy disk, you must specify the root directory.***

- **Browse**—This button lets you browse for a target path on disk.
- **Install Run-Time Engine**—This checkbox lets you include the LabWindows/CVI Run-time Engine and associated files in your distribution kit. If you know that the Run-time Engine is already on the target machine, or if you want to copy and distribute the Run-time Engine separately, you do not have to include the Run-time Engine files in your distribution kit.
- **Install Low-Level Support Driver**—This option appears in Windows 95/NT. It lets you choose whether to install the LabWindows/CVI low-level support driver on the user's

computer. The Utility Library functions shown in Table 3-1 require the LabWindows/CVI low-level driver.

**Table 3-1.** Platforms Where Utility Functions Require the Low-Level Support Driver

| Function | Platforms That Require the Low-Level Support Driver |
|---|---|
| `inp` | Windows NT |
| `inpw` | Windows NT |
| `outp` | Windows NT |
| `outpw` | Windows NT |
| `ReadFromPhysicalMemory` | Windows 95/NT |
| `ReadFromPhysicalMemoryEx` | Windows 95/NT |
| `WriteToPhysicalMemory` | Windows 95/NT |
| `WriteToPhysicalMemoryEx` | Windows 95/NT |
| `MapPhysicalMemory` | Windows 95/NT |
| `UnMapPhysicalMemory` | Windows 95/NT |
| `DisableInterrupts` | Windows 95 |
| `EnableInterrupts` | Windows 95 |
| `DisableTaskSwitching` | Windows 95 |

The LabWindows/CVI development environment and the LabWindows/CVI run-time engine each load the low-level support driver automatically at startup if it is present on disk. Under Windows 95, the name of the driver is `cvi95vxd.vxd`, and the distribution kit installs it in Windows system directory. Under Windows NT, the name of the driver is `cvintdrv.sys`, and the distribution kit installs it in the Windows `system32\drivers` directory. The distribution kit makes a registry entry for the driver under Windows NT. Refer to Chapter 5, *Creating and Distributing Standalone Executables and DLLs*, of the *LabWindows/CVI Programmer Reference Manual*, for the details of the registry entry.

- **Media Size**—The media size of your distribution diskettes. When you choose a floppy drive as your target path, LabWindows/CVI determines the media size automatically and makes this control inactive.
- **kBytes Reserved on First Disk**—Allows you to reserve space on the first disk of your distribution kit for extra files.
- **Installation Language**—The language that the installation program uses for text during the installation.

## File Groups Section

- **File Groups**—This list box lets you separate the files in your distribution kit into groups. You must assign a destination directory to each group. The installation program 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.
- **Group Destination**—This ring control sets the root destination directory for the selected group.
- **Relative Path**—This control lets you assign a relative path, based on the root destination directory, in which to install the selected group.
- **Install Program Manager Icons**—This checkbox lets you choose whether to create a Windows program group that contains icons for files in the selected file group. The installation program can install the embedded icons for `.exe` files and the default icons for `.pif`, `.com`, `.txt`, `.wri`, `.bat`, and `.hlp` files.
- **Distribute Objects and Libraries for All Compilers**—This checkbox appears in LabWindows/CVI for Windows 95/NT to help you distribute object files, static libraries, and DLL import libraries for all the compatible external compilers. When enabled, this option affects all the `.obj` and `.lib` files listed in the selected file group. LabWindows/CVI includes four 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`, `borland`, `watcom`, and `symantec`. For example, if you specify the file `c:\myapp\distr\big.lib` in a file group and the Distribute Objects and Libraries for All Compilers option is enabled, then 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
  c:\myapp\distr\watcom\big.lib
  c:\myapp\distr\symantec\big.lib
  ```

  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.

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 have to install two import libraries: one compatible with Visual C/C++ and the other with Borland C/C++. The two import libraries must be installed in the*** `msc` ***and*** `bc` ***subdirectories under the*** VXI*plug&play* `lib` ***directory. LabWindows/CVI sets this up automatically for you if you use the*** **Create DLL Project** ***command in the Function Tree Editor window with the*** **VXIplug&play Style** ***command enabled.***

- **Replace Existing Files**—This control lets you configure the installation program to always replace, never replace, replace if newer, or ask before replacing existing files.

  The Check Version option applies to files with a Windows version resource (in other words, DLLs and executables). The installation program checks the file on the distribution kit and the existing file for a version resource. If each has a version resource, the installation program replaces the existing file if the version number of the file in the distribution kit is newer than the version number in the existing file. If the file in the distribution kit has a version resource but the existing file does not, the installation program replaces the existing file. When the file in the distribution kit does not have a version resource, the installation program replaces the existing file if the date of the file in the distribution kit is newer.
- **Add Group**—This button lets you add a new group to your distribution kit.
- **Edit Group**—This button lets you edit the selected group.
- **Delete Group**—This button lets you delete the selected group from your distribution kit.

## Main Section

- **Default**—Resets all controls in the Create Distribution Kit dialog box to their default values. When you create a new distribution kit, you can click on **Default** to undo changes you have made to the controls 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 on*** **Default** ***in error, click on*** **Cancel** ***to prevent this change to your distribution kit.***

- **Build**—This button lets you build your distribution kit.
- **Cancel**—This button lets you cancel the **Create Distribution Kit** operation.
- **Advanced**—When you click on this button, the Advanced Distribution Kit Options dialog box appears.

## Advanced Distribution Kit Options

Click on the **Advanced** button in the Create Distribution Kit dialog box to open the Advanced Distribution Kit Options dialog box, as shown in Figure 3-14.

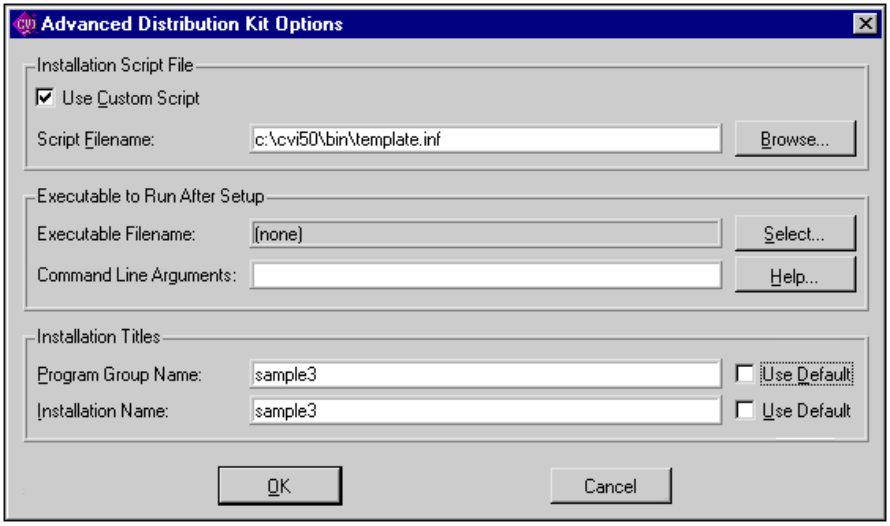


**Figure 3-14.** Advanced Distribution Kit Options Dialog Box

## Installation Script File Section

- **Use Custom Script**—Select this option to enter the name of a customized installation script file for your distribution kit. The default installation script file is `cvi\bin\template.inf`.

  The installation script for distributing VXI*plug&play* instrument drivers is `cvi\bin\vxipnp.inf`. The instructions for using the VXI*plug&play* installation script are in the file `cvi\bin\vxipnp.doc`. If you use the **Create DLL Project** command in the Function Tree Editor Window with the **VXIplug&play Style** command enabled, LabWindows/CVI automatically generates project settings so that the **Create Distribution Kit** command uses the VXI*plug&play* installation script and distributes all the files required of a VXI*plug&play* installation.

- **Script Filename**—The pathname of the customized installation script file. You can use the **Browse** button to select an existing filename.

### Executable to Run After Setup

- **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.
- **Command Line Arguments**—The command line arguments to pass to the executable to run after the installation is complete. Use the **Help** button to view detailed information on special macros you can use in this control.

### Installation Titles

- **Program Group Name**—The name of the program group created during the installation. If you select the Use Default option, LabWindows/CVI uses the following priority to determine the program group name.
  1. If the project target is an executable and you have entered an application title in the Create Standalone Executable dialog box, LabWindows/CVI uses the application title.
  2. Otherwise, if you have created the target executable, DLL, or static library, LabWindows/CVI uses the base file name of the target.
  3. 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 Program Group Name.

## Run Menu

This section explains how to use the commands in the Project window **Run** menu, as shown in Figure 3-15.

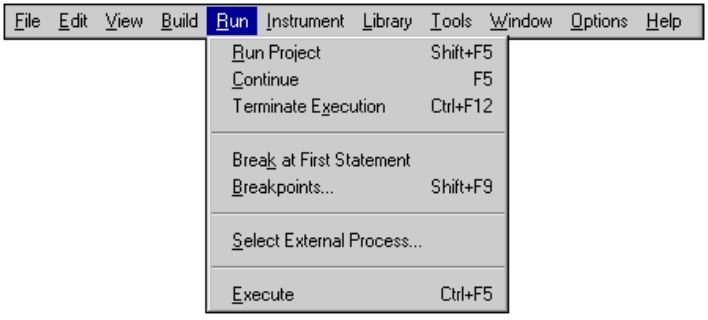


**Figure 3-15.** Run Menu

You can use commands in the **Run** menu to run your program and assign breakpoints. For more information about breakpoints, refer to the *Introduction to Breakpoints and Watch Expressions* section in Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*.

## Run Project

The **Run Project** command compiles all source files you have modified or marked for compilation, links them together, and runs the project. If LabWindows/CVI encounters any build errors, the command terminates and the Build Errors window appears with a list of errors.

### Run-Time Error Reporting

During the execution of a program, LabWindows/CVI can report various run-time errors. One example of a run-time error is a call to a LabWindows/CVI library function in which an array or string is not large enough to hold the output data.

When such errors occur, a dialog box appears identifying the type of error and the location in the program where the error occurred. Then LabWindows/CVI lists the error in the Run-Time Errors window.

LabWindows/CVI then suspends the program so you can inspect the values of variables in the Variables window. To terminate a program that has been suspended because of a run-time error, select the **Terminate Execution** command or press <Ctrl-Alt-SysRq> under Windows 3.1 or <Ctrl-F12> under Windows 95/NT, and <Control-F12> under UNIX.

## Continue

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

## Terminate Execution

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

## Break at First Statement

Select **Break at First Statement** to force LabWindows/CVI to break a program on the first executable statement. When activated, this command has a checkmark beside it in the menu.

## Breakpoints. . .

The **Breakpoints** command opens the Breakpoints dialog box that contains a list of the breakpoints in the project. You can add, delete, or edit project breakpoints from this dialog box. For a complete description of this command, refer to the *Run Menu* section of Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*.

## Select External Process. . .

This command applies only when you set the **Target** item in the **Build** menu to **Dynamic Link Library**. The **Select External Process** command allows you to specify a standalone 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 **Run *xxx*.exe**, where *xxx*.exe is the filename of the external program. When you execute the **Run *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 project.

## Execute

The **Execute** command launches the executable that you last created from the Create Standalone Executable dialog box. If you have not yet created an executable file, or the current target type for the project is DLL or Static Library, then this command dims.

# Using Instrument Drivers

This section presents a general overview of instrument drivers. For detailed information on creating instrument drivers, refer to the *LabWindows/CVI Instrument Driver Developers Guide*.

Load and unload instrument drivers using the commands in the **Instrument** menu, as shown in Figure 3-16.

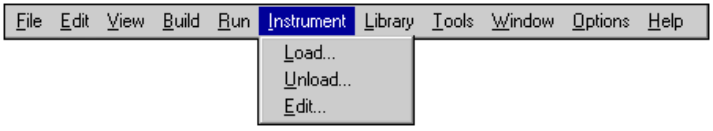


**Figure 3-16.** Instrument Menu

An *instrument driver* is a set of high-level functions with graphical function panels that make programming easier. It encapsulates many low-level operations, such as data formatting and communication with GPIB, RS-232, and VXI, into intuitive, high-level functions. An instrument driver usually controls a physical instrument, but it also can be 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 three or four files. Each file has the same base filename, which is 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.

- The function panels are in a file with the extension `.fp`. Refer to Chapter 5, *Using Function Panels*, for a detailed description of function panels.
- 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 an include file with the extension `.h`.
- The instrument driver program can be in one of several different types of files.
  - A source file with the extension `.c`.
  - An object file that contains one or more compiled C modules with the extension `.obj` under Windows, or `.o` under UNIX, or a library file with the extension `.lib` under Windows or `.a` under UNIX. The compilation must be done by LabWindows/CVI or a compatible external compiler. Refer to the *LabWindows/CVI Programmer Reference Manual* for more information on compatible external compilers.
  - A dynamic-link library, `.dll` under Windows 3.1 only. Under Windows 95/NT, each DLL must be accompanied by an import library (`.lib`) file. If you want to use a DLL instrument driver, the import library is the instrument driver program file.
  - A path (`.pth`) file used to specify a `.dll` that is not in the same directory as the `.fp` file, under Windows 3.1 only.

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` (`.o`), or `.lib` (`.a`) file. The presence of the `.h` file is essential, because you must include it in your program in order 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 include (`.h`) file in the same directory as the `.fp` file. The installation program places it in the `include` subdirectory under the VXI*plug&play* directory. LabWindows/CVI can find the include files in the VXI*plug&play* include directory.

When you install a VXI*plug&play* instrument driver, the installation program places the source (`.c`) file in the same directory as the `.fp` file. Under Windows, the installation program also 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 `msc` and `bc` subdirectories in the VXI*plug&play* `lib` directory. The DLL is in the VXI*plug&play* `bin` directory. The installation program adds the VXI*plug&play* `bin` directory to the `PATH` environment variable so that the DLL can be found using the standard Windows DLL search algorithm.

Under Windows 95/NT, if the `.fp` file is under the VXI*plug&play* framework directory and your current compatible compiler is Visual C/C++ or Borland C/C++, LabWindows/CVI can find the appropriate import library. If your current compatible compiler is Watcom C/++ or Symantec C/C++, LabWindows/CVI looks for the import library in the `wc` or `sc` subdirectory in the VXI*plug&play* `lib` directory. However, import libraries for these two compilers do not normally accompany VXI*plug&play* instrument drivers. If LabWindows/CVI finds the import library, it gives it precedence over the `.c` file as the program file for the instrument driver.

Under Windows 3.1, the import library is 16-bit; LabWindows/CVI cannot use it. LabWindows/CVI uses the program file in the directory of the `.fp` file. If LabWindows/CVI cannot find a program file in the directory of the `.fp` file, it searches for a DLL using the standard Windows search algorithm. It performs this search whether or not the `.fp` file is under the VXI*plug&play* framework directory.

## Loading/Unloading Instrument Drivers

You can load and unload instrument drivers manually using the **Instrument** menu. Instrument drivers loaded through the **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 can incorporate instrument drivers into the project using the **Add to Project** command in the **File** menu of a Function Panel window or the Function Tree Editor window, or in the **Edit** menu of the Project window. The `.fp` file represents the instrument driver in the project list. If the `.fp` file is in the project list when you open the project, LabWindows/CVI automatically loads the instrument driver and removes it when you unload 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. This can

occur when you obtain the source code for the instrument driver and then compile it. 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. *x*`.obj` (*x*`.o` under UNIX) and *x*`.c` thus cannot be in the project list at the same time unless you exclude one or both.
- If you are working in Windows 95/NT and the following conditions apply, 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 listed in the following table,

  Then LabWindows/CVI loads the `.lib` file. The corresponding subdirectories appear in Table 3-2.

**Table 3-2.** VXI*plug&play* Framework Subdirectories

| Compatible Compiler | Corresponding Subdirectory That Contains the .lib File |
|---|---|
| Visual C/C++ | `lib\msc` |
| Borland C/C++ | `lib\bc` |
| Watcom C/C++ | `lib\wc` |
| Symantec C/C++ | `lib\sc` |

- 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` under Windows or `.a` under UNIX
  2. `.obj` under Windows or `.o` under UNIX
  3. `.pth` under Windows 3.1 only
  4. `.dll` under Windows 3.1 only
  5. `.c`

**Note** ***In Windows 3.1, if the*** `.pth` ***file supplies a simple path to the*** `.dll` ***file, LabWindows/CVI loads the*** `.dll` ***file using the standard Windows DLL search algorithm.***

- If you are working in Windows 3.1 and no program file has been found, LabWindows/CVI searches for a DLL with the same base name as the `.fp` file using the standard Windows search algorithm.

### 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 **Instrument** menu.

This is useful if you want to use `.fp` files for documenting 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 when no program is associated with the instrument, LabWindows/CVI reports a run-time error. It is possible to associate a `.c` file with a `.fp` file after you load the `.fp` file. Refer to the description of the **Edit** command for more information.

## 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 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 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, by selecting **Instrument»Edit**, or by 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 **Option»Edit Function Panel Window** from a Function Panel window, by selecting **Edit»Edit Function Panel Window** from a Function Tree Editor window, or by 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**, by selecting **Tools»Go To Definition** from a Function Panel Editor window, or by selecting **Go To Definition** from the context menu in the Function Tree Editor window.
- 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 window.

# Instrument Menu

The **Instrument** menu is a *dynamic* menu. It 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.

For example, after you load two instruments, the **Instrument** menu appears, as shown in Figure 3-17.

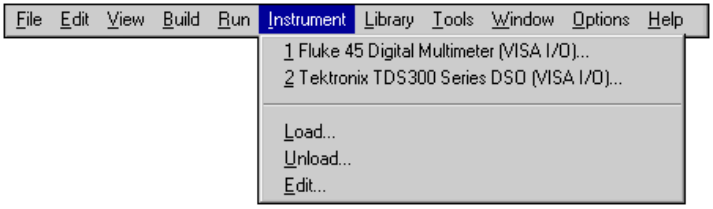


**Figure 3-17.** Instrument Menu with Two Instruments Loaded

## Load. . .

When you select the **Load** command, a dialog box appears that is similar to the dialog box displayed by the **Open** command in the **File** menu. In the Instrument Load dialog box, the filename `*.fp` appears in the text box. 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 for a program file with the same base filename in the same directory. If it finds one, it loads the instrument driver program along with the function panels.

For VXI*plug&play* instrument drivers, the program file can be in a different directory. Refer to the *Precedence Rules for Loading the Instrument Driver Program File* section earlier in this chapter.

### File Format Conversion

If the `.fp` file you are loading was created using 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 subsequently to save the converted `.fp` file to disk.

## Unload. . .

When you select the **Unload** command, a dialog box appears that contains a scrollable list of all the instruments you loaded with the **Load** menu. From this dialog box you can select one or more instrument drivers to unload.

## Edit. . .

You can use the **Edit** command to edit an instrument driver program in a Source window or an instrument driver function tree in a Function Tree Editor window. When you select the **Edit** command, the Edit Instrument dialog box shown in Figure 3-18 appears.

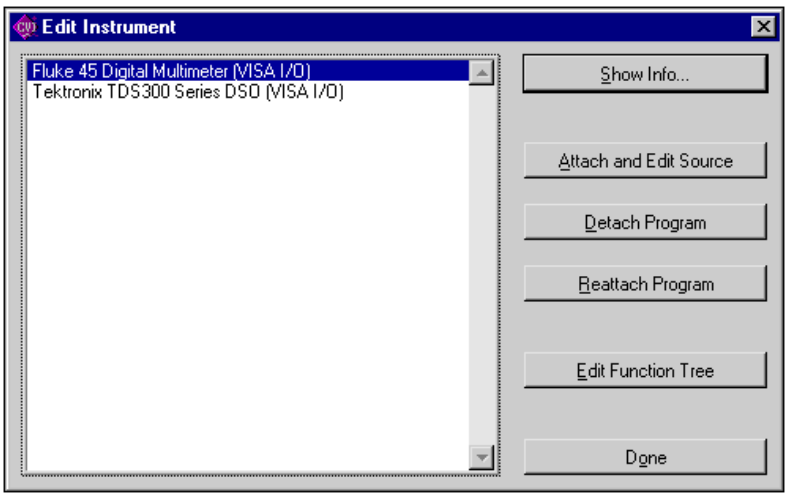


**Figure 3-18.** Edit Instrument Dialog Box

The dialog box displays instrument drivers you loaded as part of the project or through the **Instrument** menu. The commands in the Edit Instrument dialog box are as follows.

- **Show Info**—Opens the read-only Instrument Driver Information dialog box, as shown in Figure 3-19.

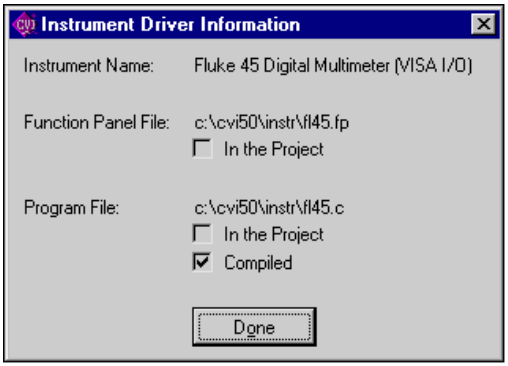


**Figure 3-19.** Instrument Driver Dialog Box

- **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.
- **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 the precedence rules outlined in the *Precedence Rules for Loading the Instrument Driver Program File* section earlier in this chapter.
- **Edit Function Tree**—A Function Tree Editor window appears that displays the function tree for the selected `.fp` file.

## Accessing Function Panels from the Instrument Menu

When you select an instrument name in the **Instrument** menu, the Select Function Panel dialog box appears, as shown in Figure 3-20.

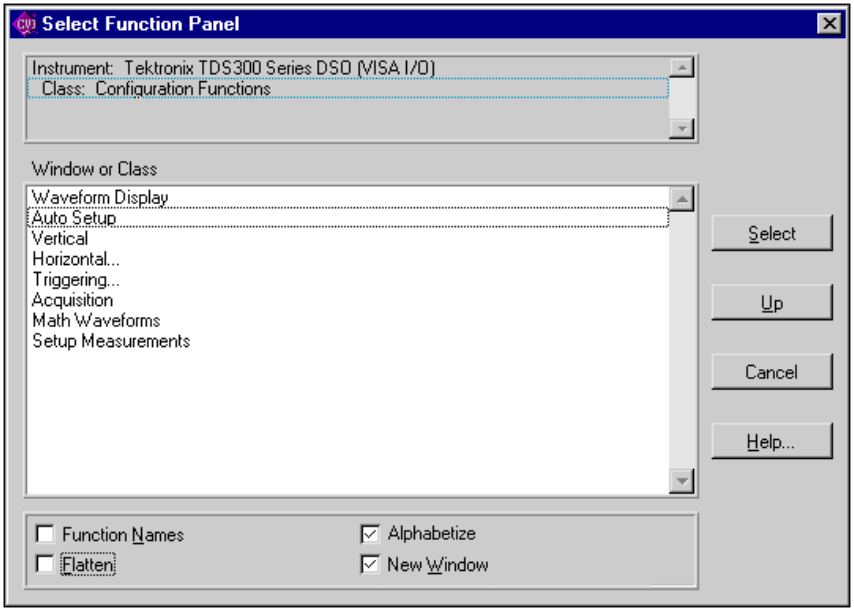


**Figure 3-20.** Select Function Panel Dialog Box

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 at or below the current level.

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 New Window is selected, the next selected function panel appears in a new window; otherwise, LabWindows/CVI overwrites the current Function Panel window. This overrides the Use Only One Function Panel Window option in the **Environment** command of the **Options** menu.

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 on 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 5, *Using Function Panels*, for more information about function panels.

To close the dialog box, select **Cancel**.

The dialog box contains a **Help** button. You can get help information about the functions and classes listed in the dialog box. Select **Help** or press <F1> to display the Function, Window, or Class Help dialog box, as shown in Figure 3-21.

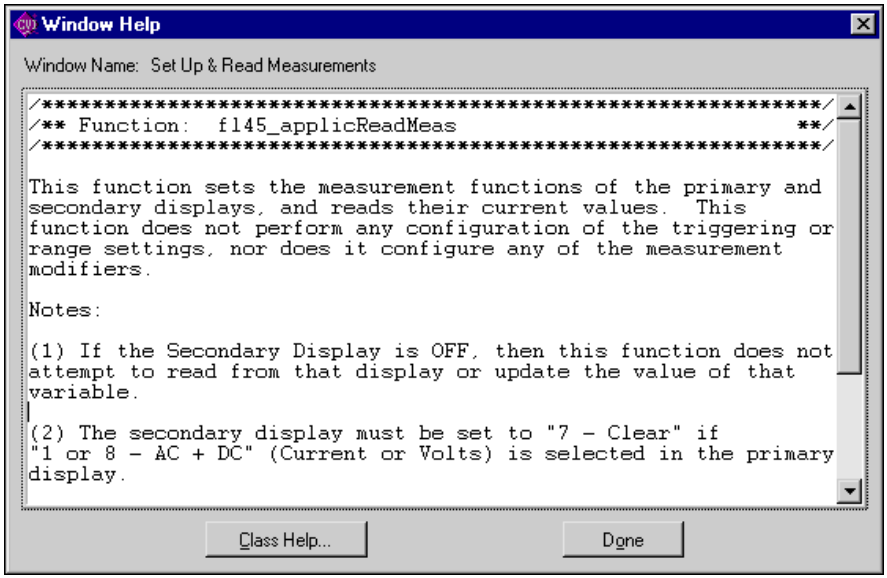


**Figure 3-21.** Instrument Help Dialog Box

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

# Library Menu

This section explains how to use the commands in the **Library** menu for the Project window.

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.

Figure 3-22 shows the **Library** menu.

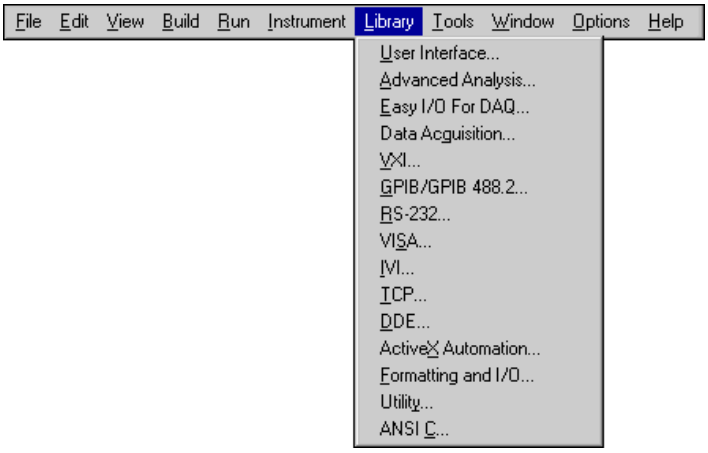


**Figure 3-22.** Library Menu

When you select a library name in the **Library** menu, you can access the library function panels in the same way you access instrument driver function panels when you select instrument names in the **Instrument** menu. For more information, refer to the *Accessing Function Panels* section in Chapter 5, *Using Function Panels*.

## User Interface. . .

The User Interface Library is a set of functions for controlling a graphical user interface. A graphical user interface is a collection of objects such as menu bars, panels, controls, and graphs. You can construct the user interface programmatically or with the User Interface Editor. Refer to the *LabWindows/CVI User Interface Reference Manual* for more information on creating and controlling a user interface.

## Analysis. . ./Advanced Analysis. . .

The Analysis Library includes functions for one-dimensional (1D) and two-dimensional (2D) array manipulation, complex operations, matrix operations, and statistics. Refer to Chapter 3, *Analysis Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for analysis function descriptions.

The Advanced Analysis Library is an optional package. It includes additional functions for signal generation, signal processing, and curve fitting. Refer to the *LabWindows/CVI Advanced Analysis Library Reference Manual* for advanced analysis function descriptions.

## Easy I/O For DAQ. . . (Windows Only)

The Easy I/O for DAQ Library contains functions which make writing simple DAQ programs easier than if you use the Data Acquisition Library. For detailed descriptions of the Easy I/O for DAQ functions, refer to Chapter 10, *Easy I/O for DAQ Library*, of the *LabWindows/CVI Standard Libraries Reference Manual*.

## Data Acquisition. . . (Windows Only)

Use the Data Acquisition Library to control your National Instruments data acquisition boards. Refer to the *NI-DAQ User Manual for PC Compatibles* and the *NI-DAQ Function Reference Manual for PC Compatibles* included with your data acquisition hardware, for a software overview and function descriptions.

## VXI. . .

Use the VXI Library to control your VXI instruments from a National Instruments embedded VXI controller or a PC equipped with a MXI controller. The VXI Library includes functions for Commander and Servant Word Serial Protocol, low-level VXIbus access, local resource access, VXI signals, interrupts, triggers, system interrupt handlers, and system configuration. Refer to the *NI-VXI User Manual* and the *NI-VXI Programmer Reference Manual*, included with your LabWindows/CVI VXI Development system, for a software overview and function descriptions.

## GPIB/GPIB 488.2. . .

Use the GPIB Library to communicate with GPIB instruments over the NI-488/488.2 protocol. You must have a National Instruments GPIB hardware interface and software driver. Refer to the *NI-488.2M Software Reference Manual* and either the *NI-488.2 Function Reference Manual for DOS/Windows or the NI-488.2M Function Reference Manual for Win32* included with your GPIB 488.2 controller board, for a software overview and function descriptions. Refer to Chapter 4, *GPIB/GPIB-488.2 Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for a description of the Device Manager routines, which ensure against device name conflicts when using instrument drivers.

## RS-232. . .

Use the RS-232 Library to control the RS-232 serial ports on your computer. Refer to Chapter 5, *RS-232 Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## VISA. . .

The VISA Library gives VXI and GPIB software developers, particularly instrument driver developers, a single interface library for controlling VXI, GPIB, and RS-232 instruments. The *NI-VISA Programmer Reference Manual*, available upon request, describes the functions.

## IVI. . .

The IVI Library gives developers a structured framework for creating VXI*plug&play* instrument drivers with advanced features such as state caching, simulation, and compatibility with generic instrument classes. The IVI wizard supplements the library, automatically creating the skeleton of an IVI driver that includes source code and function panels. The *LabWindows/CVI Instrument Driver Developers Guide* contains the IVI Library function reference and instructions on how to create IVI drivers. Use the **Create IVI Instrument Driver** command in the **Tools** menu to start the IVI wizard.

## TCP. . .

Use the Transmission Control Protocol (TCP) Library to communicate over TCP networks. Refer to Chapter 7, *TCP Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## X Property. . . (UNIX Only)

Use the X Client Property Library to communicate among X clients by reading and writing properties to and from X Windows. Refer to Chapter 9, *X Property Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## DDE. . . (Windows Only)

Use the Dynamic Data Exchange (DDE) Library to create an interface with other Windows applications using the DDE standard. Refer to Chapter 6, *DDE Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## ActiveX Automation. . . (Windows 95/NT Only)

Use the ActiveX Automation Library in conjunction with the ActiveX Automation Controller wizard to control Automation servers. Invoke the wizard using the **Create ActiveX Automation Controller** command in the **Tools** menu. The wizard allows you to browse an Automation server and select which methods you want to use. It then generates an instrument driver with a C API and function panels for using the selected methods. The ActiveX Automation Library contains low-level functions that the generated instrument drivers use and high-level functions that you use to manipulate the data you pass to and

from the instrument driver functions. Refer to Chapter 11, *ActiveX Automation Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## Formatting and I/O. . .

Use the Formatting and I/O Library functions to input and output data to files and manipulate the format of data in a program. Refer to Chapter 2, *Formatting and I/O Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## Utility. . .

The Utility Library contains functions that do not fit into any of the other LabWindows/CVI libraries. You can use these functions for file manipulation and other miscellaneous tasks. Refer to Chapter 8, *Utility Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for function descriptions.

## ANSI C. . .

The ANSI C Library contains the ANSI C functions available in LabWindows/CVI. Refer to Chapter 1, *ANSI C Library*, in the *LabWindows/CVI Standard Libraries Reference Manual* for more information.

## 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. Refer to the *Instrument Driver Files* section earlier in this chapter for more information on loading files with the **Instrument** menu. The main difference between modules you load as instrument drivers and those you load as user libraries is that you can unload instrument drivers with the **Unload** command in the **Instrument** menu, 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 detailed information about writing an instrument driver.

You install user libraries with the **Library Options** command in the **Options** menu. The next time you enter 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

If you are running under Windows, 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.

If you are running under Windows 95/NT, 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 to only a subset. Refer to the *Calling Windows SDK Functions* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual.*

Under UNIX, you can call functions in the host system libraries. LabWindows/CVI automatically loads certain Sun system libraries. For more information on using UNIX system functions, refer to the *Calling Sun C Library Functions* section in Chapter 5, *UNIX Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual.*

# Tools Menu

This section explains how to use the commands in the Project window **Tools** menu, as shown in Figure 3-23.

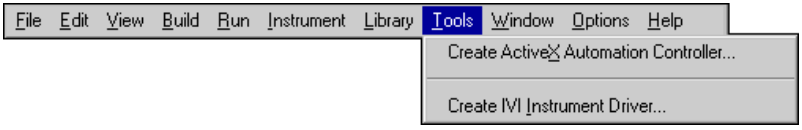


**Figure 3-23.** Tools Menu

## Create ActiveX Automation Controller. . . (Windows 95/NT Only)

Use the **Create ActiveX Automation Controller** command to generate a new instrument driver for an ActiveX Automation server. When you select the **Create ActiveX Automation Controller** command, the Select ActiveX Automation Server dialog box appears, as shown in Figure 3-24.

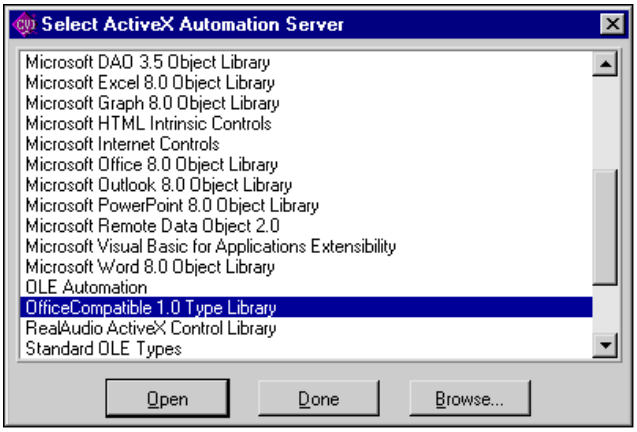


**Figure 3-24.** Select ActiveX Automation Server Dialog Box

- **Automation Servers**—This list box contains the ActiveX Automation servers on the current machine. LabWindows/CVI extracts this list from the system registry.
- **Open**—This button opens the Browse Type Library dialog box for the currently selected server.
- **Done**—This button closes the dialog box.
- **Browse**—This button lets you browse for an ActiveX Automation server object library file on disk. When you select an ActiveX Automation server from the Browse dialog box, LabWindows/CVI adds the server to the system registry if it was not already there and opens the Browse Type Library dialog box.

## Browse Type Library Dialog Box

Use the Browse Type Library dialog box to create an instrument driver to control the ActiveX Automation server you selected. The dialog box activates a wizard that generates source code and function panels from information in the ActiveX Automation server object library.

**Note** ***The ActiveX Automation Library contains functions that you use in conjunction with the functions in the generated instrument driver.***

In the dialog box, you must select the server objects to include in the instrument driver. Refer to the server documentation to determine which objects you want to use. Many servers provide online help that you can access from the dialog box. Selecting a large number of server objects results in large source files and longer compile times.

The wizard generates various types of functions. You can get and set object properties through property functions. The wizard generates one set of property functions to apply to all server objects. You use a property constant to identify a specific property to a property function. You can use a method function to invoke a method of an object. The driver contains a separate method function for each method of each object. You can use the object creation functions to create top-level objects. After you have created a top-level object, you can create other objects using the property and methods functions. Three object creation functions exist for each top-level object.

LabWindows/CVI concatenates the instrument prefix, an object tag, and a property or method tag to create each constant name and method function name in the instrument driver. You can specify the object, method, and property tags, but the dialog box provides default values. The default value for a tag is the object name, method name, or property name that the server object library defines.

Given the size of object and method names in some servers, some of the function names suggested by the dialog box can be very long. However, the function panel file format imposes a limit of 31 characters, excluding the instrument prefix and underscore. In addition, the wizard imposes a limit of 42 characters on property constants, excluding the instrument prefix and underscore. Therefore, it might be necessary for you to edit the object, property, and method tags to make the function names and property constants smaller. To assist you, the dialog box flags names that are too long. The dialog box also automatically abbreviates object, property, and method tags for common ActiveX Automation servers, such as Microsoft Excel 8.0. If you have names that are too long, you must abbreviate object tags before abbreviating property and method tags. Because LabWindows/CVI does not include the instrument prefix in the name length limit, abbreviating it does not allow you to have longer property, method, or object tags.

Figure 3-25 shows the Browse Type Library dialog box for OfficeCompatible 1.0.

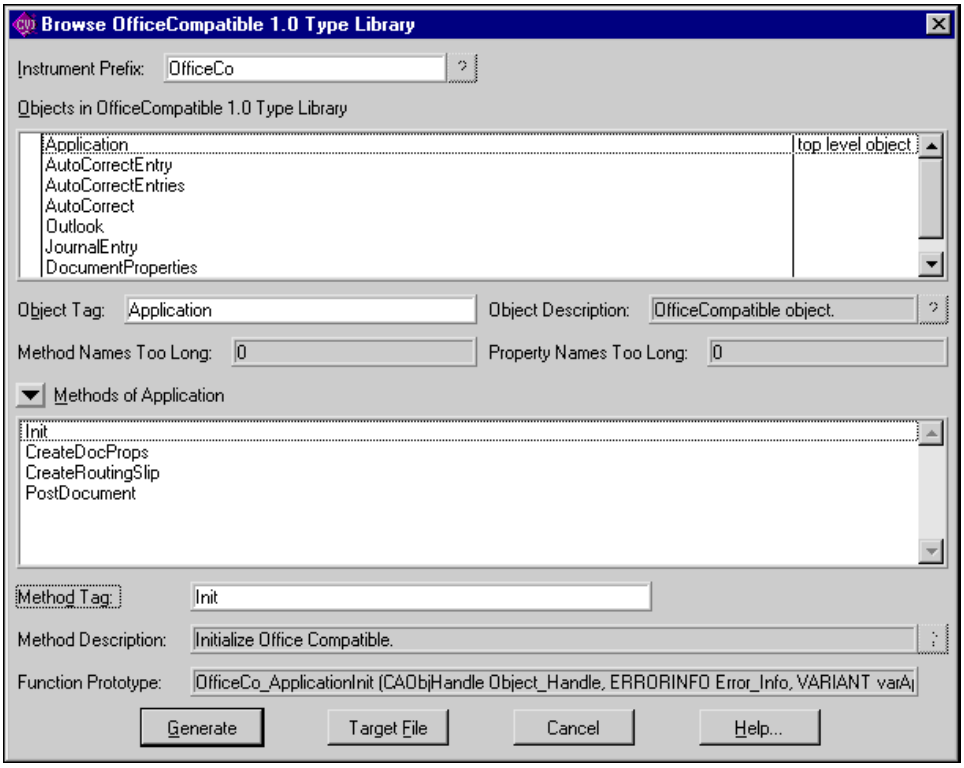


**Figure 3-25.** Browse Type Library Dialog Box for OfficeCompatible 1.0

- **Instrument Prefix**—The instrument prefix for the generated ActiveX Automation Controller instrument driver. You must either leave the control empty or enter a valid C identifier of up to 8 characters.
- **Objects in ActiveX Automation Server**—This list box displays the objects in the ActiveX Automation server. Indicate the object(s) to include in the generated instrument driver by checking the list box entries.

  The second column of this list box indicates which objects are top-level objects. For each selected top-level object, the wizard generates `New`*`Object`*, `Open`*`Object`*, and `Active`*`Object`* functions. You must use one of these functions to create an ActiveX Automation object before you can use any of the other functions in the instrument driver. You create all other objects in the ActiveX Automation server through methods and properties of the top-level objects.
- **Object Tag**—The tag associated with the currently selected object. LabWindows/CVI adds this tag to the method and property tags to create the method function and property constant names. This string must contain only valid C identifier characters.

- **Object Description**—A description of the currently selected object. Some ActiveX Automation servers do not provide this description.
- **Method Names Too Long**—The number of methods of the currently selected object with generated function names that are longer than 31 characters. LabWindows/CVI constructs a method function name by appending the Method Tag to the Object Tag. You can reduce the length of a method function name by editing the Method Tag or Object Tag.
- **Property Names Too Long**— The number of properties of the currently selected object with generated constant names that are longer than 42 characters. LabWindows/CVI constructs a property constant name by appending the Property Tag to the Object Tag. You can reduce the length of a property constant name by editing the Property Tag or Object Tag.
- **Methods/Properties Pop-up Menu Ring**—Use this ring to display the methods or properties of the selected object. The ring setting determines whether the list box below the ring displays method or property names.
- **Methods of *Object***—This box contains the method names of the currently selected object. The **`name too long`** message appears to the right of methods that yield function names that are too long.
- **Method Tag**—LabWindows/CVI appends this string to the Object Tag to construct the function name for the currently selected method. This string must contain only valid C identifier characters.
- **Method Description**—A description of the currently selected method. Some ActiveX Automation servers do not provide this description.
- **Function Prototype**—The prototype of the function to be generated for the currently selected method. This prototype does not include the function return type and calling convention.
- **Properties of *Object***—This list box contains the property names of the currently selected object. The **`name too long`** message appears to the right of properties that yield function names that are too long. Set the pop-up menu ring to Properties to see properties in the list box.
- **Property Tag**—LabWindows/CVI appends this string to the Object Tag to construct the generated name for this property. This string must contain only valid C identifier characters.
- **Property Description**—A description of the currently selected property. Some ActiveX Automation servers do not provide this description.
- **Context Sensitive Help Buttons**—These buttons, labeled with a question mark, display help for the server, object, method, or property. If the server does not provide help for an item, the corresponding context-sensitive **Help** button dims.

- **Generate**—This button invokes the wizard to generate the function panel file, source file, and header file for the controller instrument driver. After generating the driver, LabWindows/CVI compiles the source code.
- **Target File**—This button launches the Select Target File dialog box, as shown in Figure 3-26.

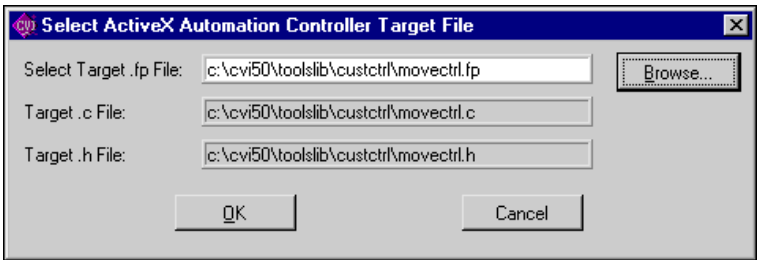


**Figure 3-26.** Select Target File Dialog Box

  - **Select Target .fp File**—The pathname of the `.fp` file into which LabWindows/CVI generates the instrument function panels. You can select either a new file or an existing file. If you select an existing, non-empty file, the wizard for the current ActiveX Automation server must have previously generated it. Refer to the *Updating Existing Controller Instrument Drivers* section below for a discussion of how the wizard operates in this case.
  - **Target .c File**—This control displays the name of the `.c` file into which LabWindows/CVI generates the instrument driver source code. LabWindows/CVI automatically creates the pathname from the pathname of the `.fp` file. You cannot modify this name.
  - **Target .h File**—This control displays the name of the header file into which LabWindows/CVI generates the instrument driver declarations. LabWindows/CVI creates this pathname automatically from the pathname of the `.fp` file. You cannot modify this name.
  - **Browse**—This button lets you browse for a target `.fp` file.
  - **OK**—This button accepts the currently displayed file names.
  - **Cancel**—This button cancels the operation, keeping the previously specified target file pathnames.
- **Cancel**—This button closes the Browse Type Library dialog box. The instrument driver is not generated.
- **Help**—This button displays help for this dialog box.

### Updating Existing Controller Instrument Drivers

You can use the Select ActiveX Automation Server dialog box to select a server for which you have already loaded an existing instrument driver. In the Browse ActiveX Automation Server dialog box, you can select additional server objects to add to the driver. The wizard adds the property constants, method functions, and creation functions for the objects to the driver.

If you select an object that was already included in the driver, the wizard gives you the option to replace the existing implementation or leave it as is.

## Create IVI Instrument Driver. . .

Use the **Create IVI Instrument Driver** command, which opens the IVI wizard, 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:

- An existing driver for a similar instrument
- The core IVI driver template
- An IVI instrument class template

The IVI 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 on the IVI wizard.

## 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. Use the **Tools Menu Options** command from the **Options** menu of the Project window to add your own entries to the **Tools** menu.

# Window Menu

You use commands in the **Window** menu, shown in Figure 3-27, to bring any open window to the front for viewing or editing.

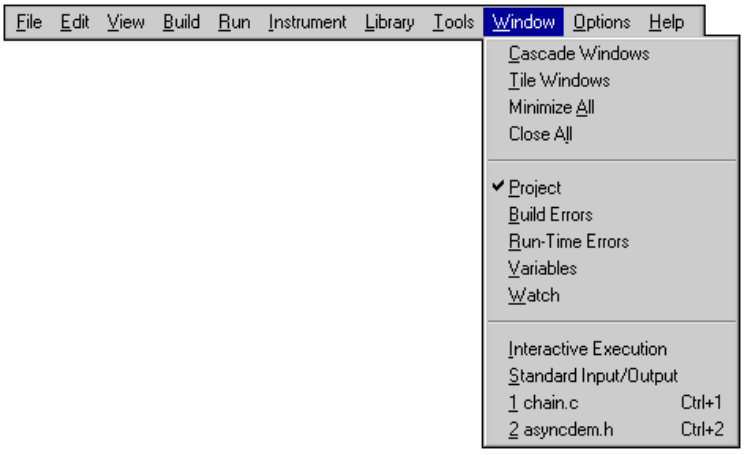


**Figure 3-27.** Window Menu

## Cascade Windows

Use this command to arrange all open windows so that each title bar is visible.

## Tile Windows

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

## Minimize All (Windows 95/NT Only)

The Minimize All command hides all the LabWindows/CVI windows, including the Project window and any User Interface Library panels displayed by a program you are currently executing. You can restore the windows by clicking on LabWindows/CVI in the Windows 95/NT task bar.

## Close All

The **Close All** command closes all the LabWindows/CVI windows, excluding the Project window and any User Interface Library panels displayed by a program you currently executing. It works on all platforms.

## Project

Use this command to bring the Project window to the front.

## 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. The **Build Errors** command in the **Windows** menu brings the Build Errors window to the front for viewing.

## 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** command in the **Windows** menu brings the Run-Time Errors window to the front for viewing.

## Variables

Use this command to bring the Variables window to the front. The Variables window shows the contents of all variables currently defined in LabWindows/CVI. 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.

Refer to Chapter 6, *Variables and Watch Windows*, for more information about the Variables 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.

Refer to Chapter 6, *Variables and Watch Windows*, for more information about the Watch window.

## Array/String Displays

If any Array/String Display windows are active, they appear in the **View** menu. Selecting one of these windows from the **View** menu brings it to the front for viewing and editing. Chapter 7, *Array and String Display Windows*, describes Array/String Display windows.

## User Interface

All open User Interface Resource (`.uir`) files dynamically appear in the **Window** menu. If the file is in the project, only the file name appears. If the file is not in the project, the full path name appears. Selecting a `.uir` file from this menu brings the file to the front in a User Interface Editor window. Refer to the *LabWindows/CVI User Interface Reference Manual* for a complete description of User Interface Editor windows.

## Function Panel

All open Function Panel windows dynamically appear in the **Window** menu. Select a Function Panel window from this menu to bring that window to the front. Refer to Chapter 5, *Using Function Panels*, for a complete description of Function Panel windows.

## Function Tree

All open Function Tree files dynamically appear in the **Window** menu. If the file is in the project, only the file name appears. If the file is not in the project, the full path name appears. Select a `.fp` file from this menu to bring that Function Tree Editor window to the front. Refer to Chapter 2, *The Function Tree Editor*, in the *LabWindows/CVI Instrument Driver Developers Guide* for a complete description of Function Tree Editor windows.

## Help Editor

All open Help Editor windows dynamically appear in the **Window** menu. Select a Help Editor window from this menu to bring that Help Editor window to the front. Refer to Chapter 4, *Adding Help Information*, in the *LabWindows/CVI Instrument Driver Developers Guide* for more information about the Help Editor 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.

Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, describes this window detail.

## Standard Input/Output

Use the **Standard Input/Output** command to bring the Standard Input/Output window to the front. Your program uses the Standard Input/Output window to print text messages and receive user input from the keyboard. You cannot edit text in the Standard Input/Output window.

Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, describes this window in detail.

## Open Source Files

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

Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, describes Source windows in detail.

# Options Menu

You use commands in the **Options** menu to set up preferences in the LabWindows/CVI environment, and execute various utilities.

This section explains how to use the commands in the Project window **Options** menu, as shown in Figure 3-28.

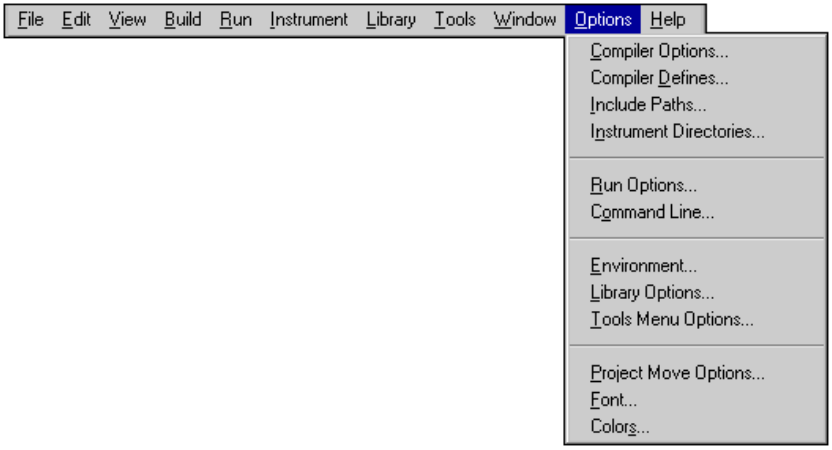


**Figure 3-28.** Options Menu

## Compiler Options. . .

You can set the LabWindows/CVI compiler options by selecting **Options»Compiler Options** in the Project window. This command opens a dialog box that allows you to set the following LabWindows/CVI compiler options:

- **Compatibility with**—(Windows 95/NT only) This option displays the current compiler compatibility mode. For more information on external compiler compatibility, refer to the *Compatibility with External Compilers* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.
- **Default Calling Convention**—(Windows 95/NT only) This option sets the compiler's default calling convention, unless the compatible compiler is Watcom. For the other compilers, the default calling convention is normally `__cdecl` but can be changed to `__stdcall`. For Watcom, it is always the stack-based calling convention. Do not change the default calling convention to `__stdcall` if you plan to generate static library or object files for all four compatible external compilers. For more information, refer to the *Calling Convention for Exported Functions* section in Chapter 3, *Windows 95/NT Compiler/Linker Issues*, of the *LabWindows/CVI Programmer Reference Manual*.

☞ **Note** ***Because you cannot set the default calling convention to*** `__stdcall` ***in the Watcom compiler or in LabWindows/CVI when in Watcom compatibility mode, you must use*** `__cdecl` ***as the default convention when you create object files, libraries, or DLLs for all four 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 Number of Compile Errors**—This option sets an upper limit on the number of compiler errors LabWindows/CVI lists in the Build Errors window for each source file.
- **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

**Caution** ***It is best to enable the* Require Function Prototypes *option. If disabled, some of the run-time error checking is also 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;
   }
}
```

- **Enable Signed/Unsigned Pointer Mismatch Warning**—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 the Enable Signed/Unsigned Pointer Mismatch Warning 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);
}
```

- **Enable Unreachable Code Warning**—This option generates a compiler warning for statements that the compiler cannot reach on execution. When you enable the Enable Unreachable Code Warning option, the LabWindows/CVI compiler generates a warning at each line of code that cannot be reached during 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;
}
```

- **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.
- **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 it, warnings can occur without being brought to your attention.
- **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. Your project compiles faster when you disable this feature.

## Compiler Defines. . .

The LabWindows/CVI compiler accepts compiler defines through the **Compiler Defines** command in the **Build** menu of the Project window.

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 your source code as a predefined macro. For example, you can use *x* as the argument to the `#if` or `#ifdef` preprocessor directive for conditional complication. If *y* contains embedded blanks, you must surround it with double quotes.

LabWindows/CVI predefines macros to help you write platform-dependent code.

- _CVI_ is defined to be 1 in LabWindows/CVI version 3.0, 301 in version 3.0.1, and 310 in version 3.1, and so on.
- _NI_mswin_ is defined if compiling under Windows 3.*x* or Windows 95/NT.
- _NI_mswin16_ is defined if compiling under Windows 3.*x*.
- _NI_mswin32_ is defined if compiling under Windows 95/NT.
- _NI_i386_ is defined if compiling on a PC.
- _NI_unix_ is defined if compiling under UNIX.
- _NI_sparc_ is defined if compiling on a Sun SPARCstation. The value of _NI_sparc_ is 1 if you are running under Solaris 1.*x* and 2 if you are running under Solaris 2.*x*.
- _CVI_DEBUG_ is defined if compiling with the debugging level set to **Standard** or **Extended**. The value of the macro is 1.

Under Windows 95/NT, LabWindows/CVI also predefines the following macros:

- _CVI_EXE_ is defined if the project target type is Standalone Executable.
- _CVI_DLL_ is defined if target type is Dynamic Link Library.
- _CVI_LIB_ is defined if target type is Static Library.
- __DEFALIGN is defined to the default structure alignment—8 for Microsoft and Symantec, 1 for Borland and Watcom.
- _NI_VC_ is defined to 220 if in Microsoft Visual C/C++ compatibility mode.
- _NI_SC_ is defined to 720 if in Symantec C/C++ compatibility mode.
- _NI_BC_ is defined to 451 if in Borland C/C++ mode.
- _NI_WC_ is defined to 1050 if in Watcom C/C++ mode.
- _WINDOWS.
- WIN32.
- _WIN32.
- __WIN32__.
- __NT__.
- _M_IX86 is defined to 400.
- __FLAT__ is defined to 1.

☞ **Note** ***LabWindows/CVI does not define*** _MSC_VER***,*** __BORLANDC__***,*** __WATCOMC__***, and*** __SC__***. 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.***

Under Windows 95/NT, the default compiler defines string contains the following definition:

```
/DWIN32_MEAN_AND_LEAN
```

This definition reduces the time and memory taken when compiling Windows SDK include files. For more information, refer to Chapter 3, *Windows 95/NT Compiler/Linker Issues*, in the *LabWindows/CVI Programmer Reference Manual*.

For all platforms, the Compiler Defines dialog box contains a list of the macros that LabWindows/CVI predefines. This list includes the name and value of each predefined macro.

## Include Paths. . .

The **Include Paths** command invokes a dialog box in which you can list paths that the compiler uses when searching for header files with simple path names. The Include Paths dialog box has two lists of paths in which to search for include files. LabWindows/CVI saves the top list with the project file. LabWindows/CVI saves the bottom list from one session to another on the same machine, regardless of the project. For a description of the entire include path search precedence, refer to the *Include Paths* discussion in Chapter 1, *LabWindows/CVI Compiler*, of the *LabWindows/CVI Programmer Reference Manual*.

## Instrument Directories. . .

Use the **Instrument Directories** command to list directories that LabWindows/CVI can search for instrument drivers on which other instrument drivers depend. 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. It 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 `cvi\toolslib` directory
4. The `cvi\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 *.FP Auto-Load List* in the *Edit* section of Chapter 3, *The Function Tree Editor Menu Bar*, in the *LabWindows/CVI Instrument Driver Developers Guide*.

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.

## Run Options. . .

The **Run Options** command invokes a dialog box you use to set the following options:

- **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. Refer to the *Stack Size* section in Chapter 1, *LabWindows/CVI Compiler*, of the *LabWindows/CVI Programmer Reference Manual* for platform-specific limitations on the stack size.
- **Debugging Level**—Four debugging levels exist for the source modules in your application.
  - **None**—In this mode, source modules execute faster, but you sacrifice the ability to set breakpoints or to 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.
  - **No Run-time Checking**—In this mode, you can set breakpoints and use the Variables window, but you have no protection from run-time memory errors, and you cannot use the Break on Library Errors option.
  - **Standard**—In this mode, you can set breakpoints, use the Variables window, and use the Break on Library Errors option. You also have protection from run-time memory errors. Refer to the *User Protection* discussion in Chapter 1, *LabWindows/CVI Compiler*, of the *LabWindows/CVI Programmer Reference Manual*.
  - **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.
- **Save Changes Before Running**—You can configure LabWindows/CVI to *never* save modified files, to *always* save modified files, or to *ask* whether to save modified files before running.
- **Break on Library Errors**—When you enable this checkbox, a breakpoint occurs when any function call to a LabWindows/CVI library results in an error.
- **Hide Windows**—When you enable this checkbox, the LabWindows/CVI environment hides all its windows until execution terminates or a breakpoint occurs.
- **Check Disk Dates Before Each Run**—When you activate the this option, LabWindows/CVI automatically compares, before each execution of the project, the disk dates of all program files used to build the project with the dates LabWindows/CVI last loaded or saved them. The program notifies you of discrepancies and gives options to resolve them. This option causes a small delay before each project execution, but is useful

when you modify program files, such as DLLs, in applications other than LabWindows/CVI. You can invoke the same action manually by selecting the **Update Program Files From Disk** command in the **Build** menu.

- **Unload DLLs After Each Run (Windows 95/NT Only)**—When you activate this checkbox, LabWindows/CVI unloads DLLs after each execution of your program in the development environment. In general, it is best to enable this option.
- **Reload DLLs on Each Run (Windows 3.1 Only)**—When activated, this checkbox has the following effects:
  - If you load DLLs directly, LabWindows/CVI regenerates the DLL glue code and reloads the DLLs each time your project runs.
  - If you load DLL glue code instead of loading the DLLs directly, LabWindows/CVI reloads the glue code and the DLLs each time you run your project.

## 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 the **Select External Process** command in the **Run** menu. The same command line arguments appear when you select the **Command Line** command from the **Options** menu.

## Environment. . .

The **Environment** command invokes a dialog box you use to set the following options:

- **Sleep Policy When Not Running Program**—Each time LabWindows/CVI checks for 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 when a user program is not running. When you run a program, the User Interface Library `SetSleepPolicy` function controls sleeping. 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. If you think you might have to make an adjustment, try the different settings and observe the resulting behavior.

- **Maximum Number of Lines in Standard Input/Output Window**—You can specify a ceiling on the number of lines maintained in the Standard Input/Output window.
- **Bring Standard Input/Output Window to front whenever modified**—Activate this checkbox to make the Standard Input/Output window appear whenever functions such as `printf` or `FmtOut` append text to it.
- **Use Only One Function Panel Window**—Activate this checkbox to overwrite the current function panel window each time you select a new function panel window.
- **Use Host System's Standard I/O (UNIX Only)**—Activate this checkbox to use the host Standard Input/Output instead of the LabWindows/CVI Standard Input/Output window.
- **Warp Mouse Over Dialog Boxes (UNIX Only)**—Activate this checkbox to automatically move the mouse cursor over dialog boxes when you initially display them.

## Keyboard Options. . . (SPARCstation Only)

The **Keyboard Options** command invokes a dialog box you use to select <Control> or <Meta> as your command accelerator key. Combine the command accelerator key with other keys to produce shortcut keys. For example, if you select <Meta> as your command accelerator, the shortcut key for the **File Save** command is <Meta-S>. If you select <Control> as the command accelerator, <Meta> works the same as <Alt>, which is the key that selects menus and menu items based on the underlined letter in the name.

## Library Options. . .

You use the **Library Options** command to specify user libraries that load automatically when you start LabWindows/CVI. You also can specify which of the optional National Instruments libraries to load on startup using the **Library Options** command. The **Library Options** command invokes the Library Options dialog box, as shown in Figure 3-29.

This dialog box contains two different areas: National Instruments Libraries and User Libraries.

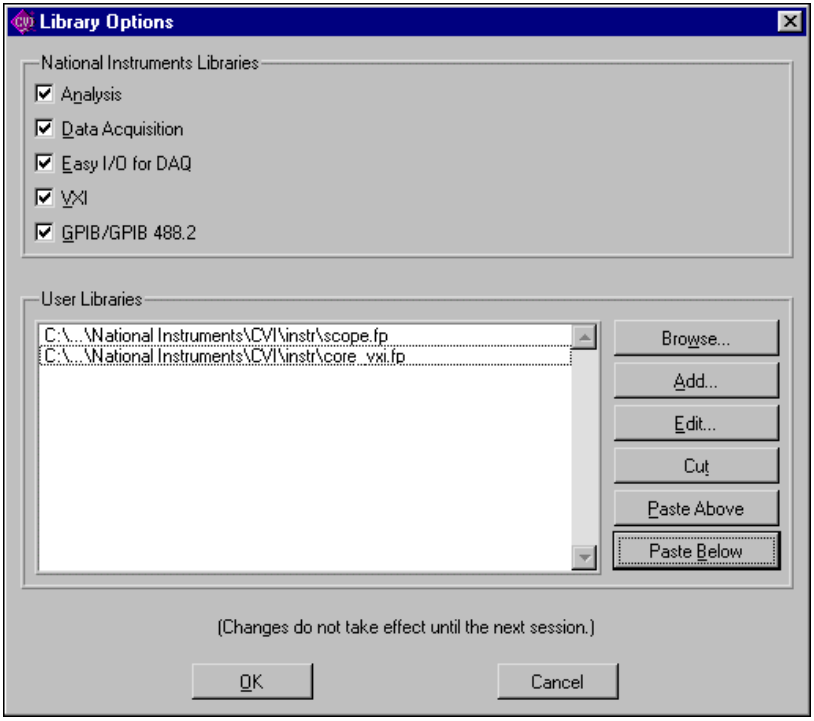


**Figure 3-29.** Library Options Dialog Box

## User Libraries

The User Libraries list contains the pathnames of user libraries that load automatically when you start LabWindows/CVI. The entries must be the pathnames of the function panel (`.fp`) files for the library modules. Because a user library has the same form as an instrument driver, anything you can load from the **Instrument** menu also can be loaded as a user library, provided it is in compiled form. Refer to the *Instrument Driver Files* discussion in the *Using Instrument Drivers* section for more information about loaded `.fp` files.

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

You specify user libraries using the **Library Options** command in the **Options** menu. LabWindows/CVI does not load the libraries or show them in the **Library** menu until the next time you launch LabWindows/CVI.

### 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.

## National Instruments Libraries

There are five optional National Instruments libraries: Advanced Analysis, Data Acquisition, Easy I/O for DAQ, VXI, and GPIB/GPIB 488.2. In the National Instruments section, click on the selection box to the left of the library name(s) that you want LabWindows/CVI to load when you start up. Use the National Instruments Libraries section of the Library Options dialog box to specify whether to load these libraries into memory when you start LabWindows/CVI. A checkmark next to the library name indicates that you want the library to be loaded. Changes do not take effect until the next time you start 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. The files in Table 3-3 belong in the `bin` directory in the LabWindows/CVI installation directory.

**Table 3-3.** Libraries in the Bin Directory of LabWindows/CVI

| Library | Windows 95/NT | Windows 3.1 | UNIX |
|---|---|---|---|
| Analysis or Advanced Analysis | `analysis.lib` | `analysis.lib` | `analysis.a` |
| Data Acquisition | `dataacq.lib` | `dataacq.obj` | `(not available)` |
| Easy I/O for DAQ | `easyio.lib` | `easyio.lib` | `(not available)` |
| VXI | `nivxi.lib` | `nivxi.lib` | `nivxi.a` |
| GPIB/GPIB 488.2 | `gpib.lib` | `gpib.obj` | `gpib.o` |

You must also install the Windows or UNIX drivers that came with your Data Acquisition, VXI, and GPIB/GPIB 488.2 hardware to use the optional libraries.

## Tools Menu Options. . .

Use the **Tools Menu Options** 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.

The **Tools Menu Options** command invokes the Tools Menu Options dialog box, as shown in Figure 3-30.

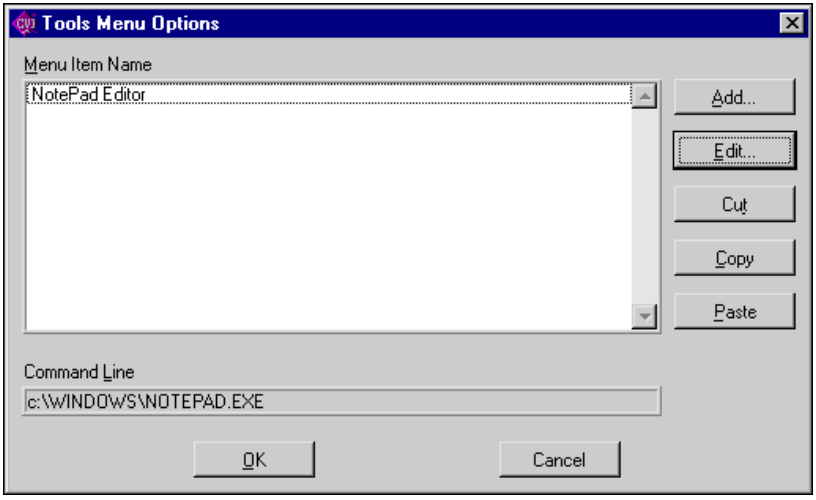


**Figure 3-30.** Tools Menu Options Dialog Box

- **Menu Item Name**—A list box that contains your current **Tools** menu entries.
- **Command Line**—An indicator that shows the command line for the currently selected menu item name.
- **Add**—Press this button to add a new entry.
- **Edit**—Press this button to edit the selected entry.

The **Add** and **Edit** buttons open the Add/Edit Tools Menu Item dialog box, as shown in Figure 3-31.

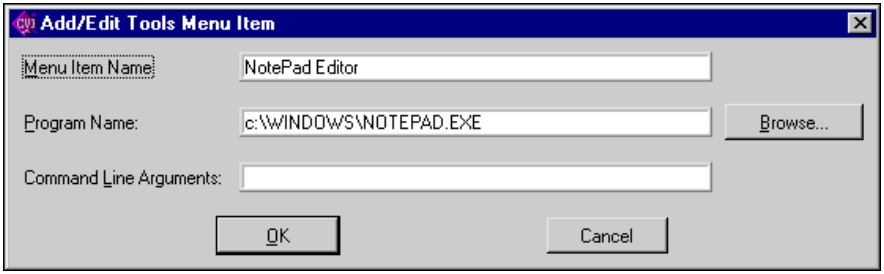


**Figure 3-31.** Add/Edit Tools Menu Item Dialog Box

- **Menu Item Name**—The string that appears in the **Tools** menu. To specify an accelerator key for the menu item, insert two underscores in front of the designated letter.
- **Program Name**—The pathname of the program to execute when you select the menu entry. You can specify full pathname, 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.

## Project Move Options. . .

This command invokes a dialog box that you use to specify what LabWindows/CVI does when you open the `.prj` file from a different directory than the one you saved it in.

If you enable the Fixup Pathnames when Project is Moved option, LabWindows/CVI updates the pathnames in the project, the project include paths, the pathnames in the distribution kit, and the pathnames of standalone executable and icon files whenever you open the project from a directory other than the one you saved it in. LabWindows/CVI assumes that the files in the project have been moved to the same position relative to the `.prj` file. If LabWindows/CVI cannot find one or more files in the project or project include paths, a dialog box appears prompting you to manually confirm those pathnames.

LabWindows/CVI might also adjust pathnames that are relative to the LabWindows/CVI directory or the VXI*plug&play* directory if the directory has changed since you last saved the project.

If `.fp` files in the project cannot be found, LabWindows/CVI searches for them in the directories listed in the Instrument Directories dialog box and in the `cvi/instr` directory.

If you disable this option, LabWindows/CVI does not update pathnames when you move the project or when the LabWindows/CVI and VXI*plug&play* directories change.

## Font. . .

Use the **Font** command to select the font and font size for the text in the Project window.

## Colors. . .

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

The **Colors** menu item opens a dialog box that contains a list box. Each line in the list box describes the purpose of the color and shows its current color state. To change the color, click and hold on the color control at the bottom left of the dialog box. A color pop-up palette appears. While holding the mouse button down, move the pointer over the desired color and then release. 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 on **Default**. All currently visible LabWindows/CVI windows immediately reflect the color changes.

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

You also can access the **Colors** command in the Source window, the Interactive Execution window, and the Standard I/O window.

The Color dialog box also contains eight color types for syntax coloring. Refer to the *Syntax Coloring* section in Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*.

When you enable the Use System Colors option, several color types associated with the Project 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 95 Display Properties dialog box.

# 4

# Source, Interactive Execution, and Standard Input/Output Windows

This chapter describes the LabWindows/CVI Source, Interactive Execution, and Standard Input/Output windows. Each of these windows supports specific tasks related to developing and executing programs.

## Source Windows

Source windows display the source code for the programs you develop. The windows behave like standard text editors. You can type text directly into a Source window or load text from an ASCII file. You can insert code from LabWindows/CVI function panels directly into Source windows. 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 254 characters in each line. A tab is one character for the purpose of line length limitation.

When you run a program in a Source window, the program must be complete and obey the syntax rules of ANSI C. Refer to the *Build Menu* and *Run Menu* sections in this chapter for more information on running programs.

## Toolbars in LabWindows/CVI

The LabWindows/CVI toolbar appears within function panels, in the Function Panel Editor window, and in Source 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.

To find out what a toolbar button does, use Toolbar Help to display the name of a toolbar button. Position the mouse cursor over that button, and either hold the cursor there for a short period of time, or click on the toolbar button with the right mouse button.

# Modifying Your Toolbars

To modify a toolbar, choose **Options»Toolbar** to display the Customize Source Window Toolbar dialog box, as shown in Figure 4-1.

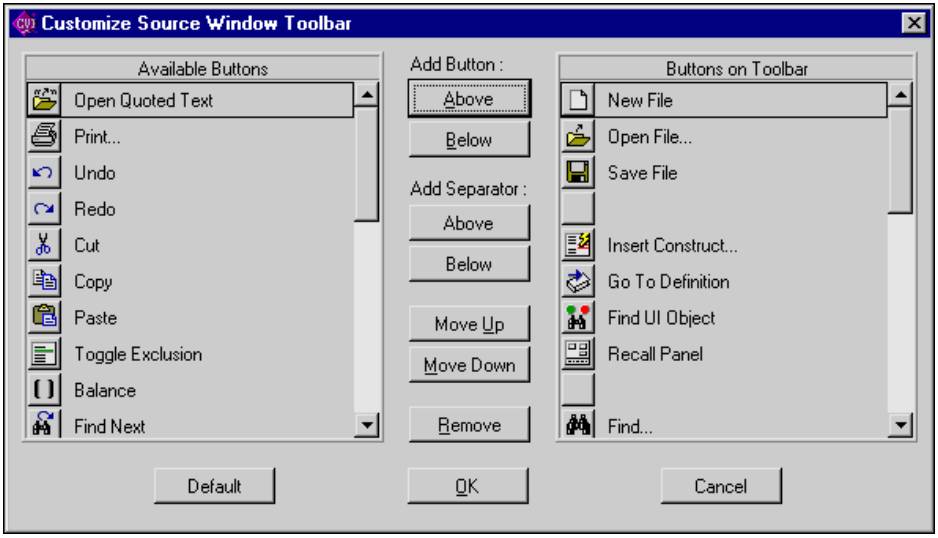


**Figure 4-1.** Customize Source Window Toolbar Dialog Box

The list box on the left of the Customize Source Window Toolbar dialog box contains names and icons of toolbar buttons that do not currently appear in the toolbar. The list box on the right contains the names and icons of toolbar buttons that currently appear in the toolbar.

# Positioning Buttons and Separators on the Toolbar

The Customize Source Window Toolbar dialog box gives you several ways to configure your 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 list box on the left. In the list box on the right, 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 select in the list box on the right. After you click on **OK**, the new item appears to the left of the other button in the toolbar. The **Below** button positions the item you add below the button you select in the list on the right. After you click on **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 list box on the right. The **Above** button adds a separator above the button you select in the list box on the right. After you click on **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 list box on the right. After you click on **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 list box on the right and use the **Remove** button to remove it from the toolbar. If you remove a button, it moves to the list box on the left. If you remove a separator, it disappears from the list. Your modifications take effect on the toolbar when you click on **OK**.

Click on **Default** to restore the default toolbar configuration for LabWindows/CVI.

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

## Notification of External Modification (Windows Only)

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. You are given the option of updating the source window from the file on disk, overwriting the file on disk with the contents of the source window, or doing nothing.

## Context Menus

You can open a context menu in the Source window by clicking the right mouse button. The context menu contains a set of the most commonly used menu commands from the Source window menu bar. The set of commands is different depending on whether the mouse is over the text editing area or over the line number or line icon area.

## Interactive Execution Window

You can execute selected portions of code in the Interactive Execution window (IEW). Unlike the Source window, you do not have to have a complete program in the IEW. For instance, you can execute C variable declarations and assignment statements without declaring a `main` function.

Use the IEW to test portions of code before you include them in your main program. Also, you can use the IEW to execute functions exported by a loaded instrument or by a file in the project, if the project has been linked. The IEW 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 IEW.

When you execute a function from a function panel, LabWindows/CVI inserts the function call into the IEW, where it executes. In this way, the IEW keeps a record of the functions you execute from function panels.

When LabWindows/CVI copies a function call from a function panel to the IEW for execution, it inserts the code after all the pre-existing lines. LabWindows/CVI also inserts an include statement for the header file associated with the function in the IEW 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 IEW. An *excluded* line appears in a different color, and the LabWindows/CVI compiler ignores it. Refer to the *Toggle Exclusion* section of this chapter for more information about excluded lines.

When you execute code in the IEW, LabWindows/CVI automatically excludes all declarations. This is why you must avoid placing executable statements on the same line as declarations in the IEW. 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 IEW remain in effect until you execute the **Clear Interactive Declarations** command from the **Build** menu or the **Clear Window** command from the **Edit** menu.

Rules for executing code in the IEW are as follows:

- When executing code from the IEW, data declarations must precede any program statements. Function declarations are also necessary unless you disable the Require Function Prototypes option in the Compiler Options dialog box of a Project window.
- You cannot include Function definitions in the IEW. 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 IEW 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 IEW:

```
extern int x=6;
```

# Standard Input/Output Window

While a program is executing, it often prints messages and data to the screen. LabWindows/CVI can display text-based messages and data in the Standard Input/Output window. LabWindows/CVI also can accept user input through the Standard Input/Output window. This window can contain up to one million lines and 254 characters per line of screen input and output. You can, however, place a lower ceiling on the number of lines. Use the **Environment** command in the **Options** menu.

You can save the contents of the window to a file, but you cannot directly edit the contents.

You can clear the Standard Input/Output window using the **Clear Window** command in the **Edit** menu or the `Cls` function in the Utility library.

# Using Subwindows

The Source, Interactive Execution, and Standard Input/Output 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 to a lower position in the window. You can then switch between the subwindows by pressing <F6> or by clicking in a subwindow with the mouse.

# 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. The selected text appears in reverse video.

To select text with the keyboard, hold down the <Shift> key 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 on the first character you want to select and drag the mouse over the remaining characters. To select a word, double-click on the word. To select a line, triple-click on 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 Figures 4-2 through 4-4.

Character Select mode highlights all characters from where you begin selecting text to where you end the selection.

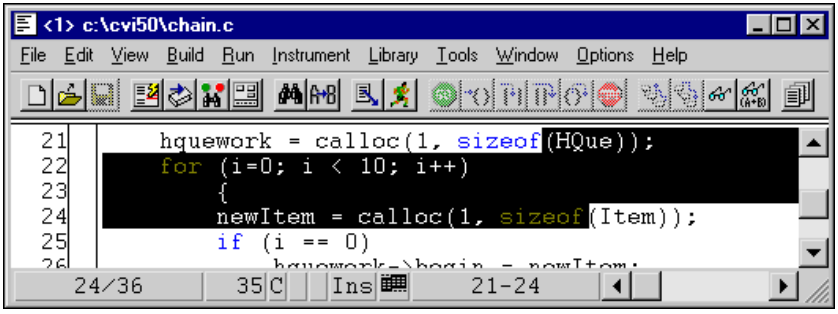


**Figure 4-2.** Selecting Text Using Character Select Mode

Line Select Mode highlights full lines of text.

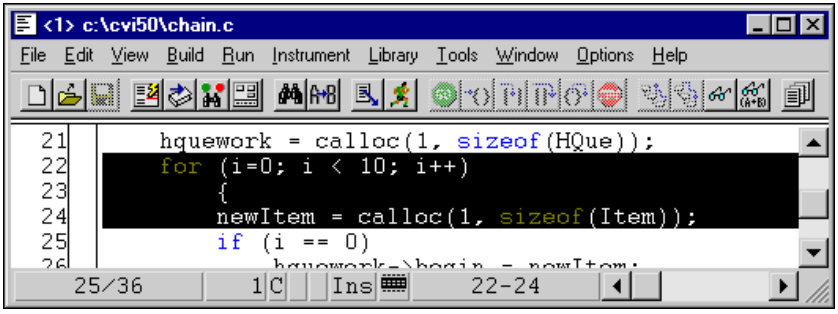


**Figure 4-3.** Selecting Text Using Line Select Mode

Column Select Mode highlights a rectangular block of text.

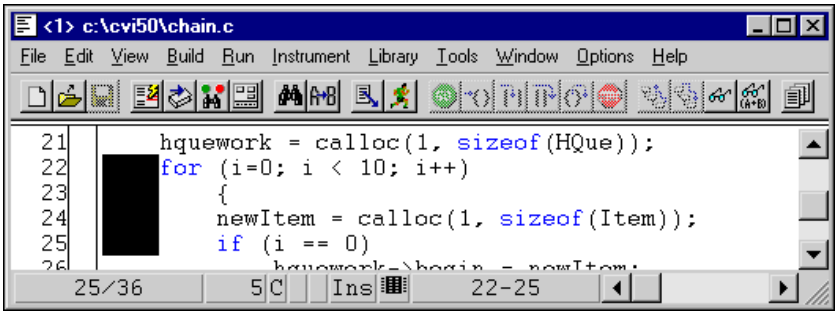


**Figure 4-4.** Selecting Text Using Column Select Mode

You can cycle through these three modes by pressing <Ctrl-Insert> on the keyboard or by clicking the mouse on the graphical icon at the bottom of the window.

# File Menu

This section contains a detailed description of the **File** menu for Source, Interactive Execution, and Standard Input/Output windows, as shown in Figure 4-5.

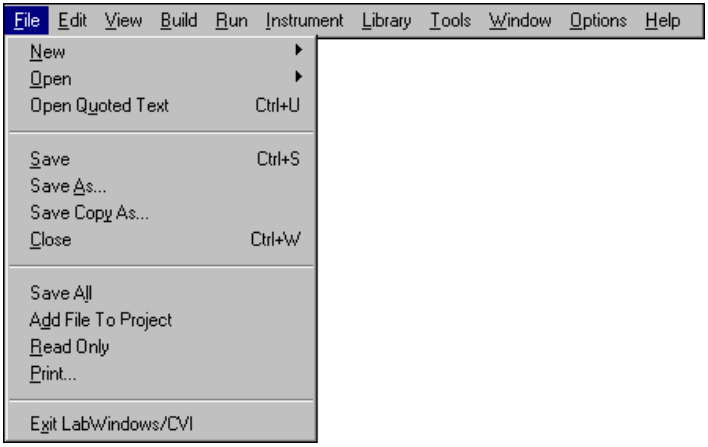


**Figure 4-5.** File Menu

## New

The **New** command operates the same way as in the Project window. For a description of this command, refer to the discussion of the **New** command in the *File Menu* section of Chapter 3, *Project Window*.

## Open

The **Open** command operates the same as in the Project window. For a description of this command, refer to the discussion of the **Open** command in the *File Menu* section of Chapter 3, *Project Window*.

## 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 proper type of window.

## Save

The **Save** command writes the contents of the active window to disk. If you want to append a different extension, type it in after the filename. If you do not want to append an extension, enter a period after the filename.

## Save As. . .

The **Save As** command writes the contents of the active window to disk using a new filename you specify and changes the name of the active window to the specified name.

## Save Copy As. . .

The **Save Copy As** command writes the contents of the active window to disk using a filename you specify *without* changing the name of the active window.

## 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 and Standard Input/Output windows. The* Hide *command visually closes the Interactive Execution and Standard Input/Output windows, but retains their contents in memory.***

## Save All

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

## Add File to Project

The **Add File to Project** command adds the file in the current window to the project list.

### Read Only

The **Read Only** command suppresses the text editing capabilities in the current window. When you initially open a file, LabWindows/CVI disables the **Read Only** command unless the file is read only on disk.

### Print. . .

The **Print** command prints the window contents to a printer or a file.

### Exit LabWindows/CVI

The **Exit LabWindows/CVI** command closes the current LabWindows/CVI session. If you have modified any open files since the last save, or if any windows contain unnamed files, LabWindows/CVI prompts you to save them to disk.

## Edit Menu

Use the commands in the **Edit** menu to edit text in Source windows and the IEW. You can copy and save text from the Standard Input/Output window, but you cannot edit it.

**Note** ***The Selecting Text in the Source and Interactive Execution Windows section earlier in this chapter describes the procedures for moving the cursor, scrolling, and selecting text.***

Figure 4-6 shows the **Edit** menu.

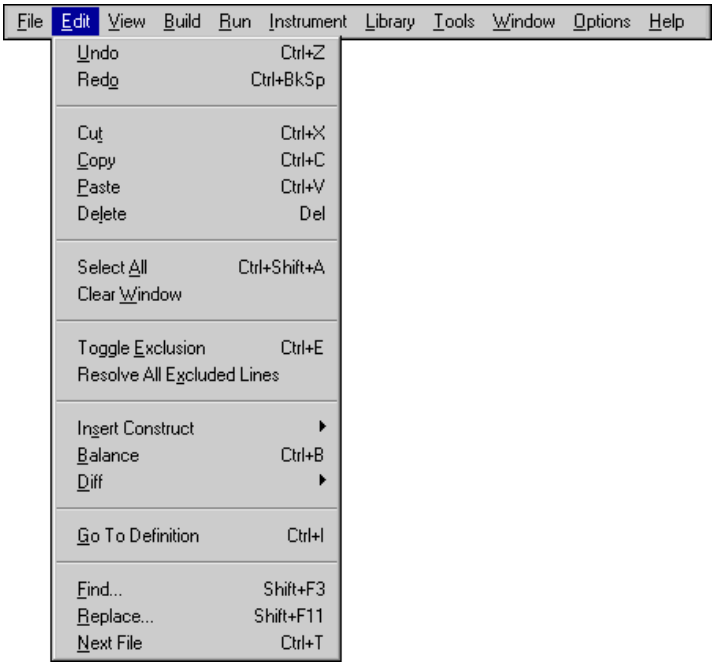


**Figure 4-6.** Edit Menu

**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 in the Clipboard. If you select an edit command while it is disabled, nothing happens.***

## Undo and Redo

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. If you go too far in using the **Undo** command, you can use **Redo** to reverse your edit actions. 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.

## Cut and Copy

To copy or cut text to the Windows Clipboard, select the text you want to place in the Clipboard and then select **Cut** or **Copy** from the **Edit** menu. LabWindows/CVI places the selected text in the Clipboard. If you used the **Copy** command, the text remains in the window. Use the **Cut** command to delete text from the window.

☞ **Note** ***The* Cut *command is disabled in the Standard Input/Output window.***

## 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 execute the **Paste** command, 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 in 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 **Paste** from the **Edit** menu.

☞ **Note** ***The* Paste *command is disabled in the Standard Input/Output window.***

## Delete

The **Delete** command deletes highlighted text without placing the text in the Clipboard.

## 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.

## Clear Window

Use the **Clear Window** command in the Interactive Execution and Standard Input/Output windows to clear the contents of these windows. The **Clear Window** command also clears any variables declared in the IEW.

☞ **Note** ***The* Clear Window *command is disabled in Source windows.***

## 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 IEW as excluded or included code. This command acts on single and multiple line selections.

You can exclude lines automatically when working in the IEW. Refer to the *Interactive Execution Window* section for more information on automatically excluding lines. Use the **Toggle Exclusion** command if you want to include these lines.

## 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.

## 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 asking you to fill in portions of the construct. You can press <Enter> or click on **OK** without filling in the controls.

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 using the **Options»Bracket Styles** command.

## 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 nest inside each other.

## Diff

Use the **Diff** command for comparing two source files to detect any differences. The **Diff** command has a submenu, as shown in Figure 4-7.

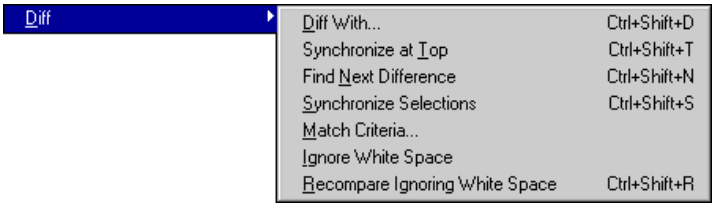


**Figure 4-7.** Diff Submenu

Use **Diff With** from a Source window 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 Selections Ignoring White Space** once a difference has been found to determine if the only difference in the selections involves white space characters.

## 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. This command does *not* work with defined constants.

## Find. . .

Use the **Find** command to locate particular text in your program. When you select the **Find** command, the Find dialog box opens, as shown in Figure 4-8.

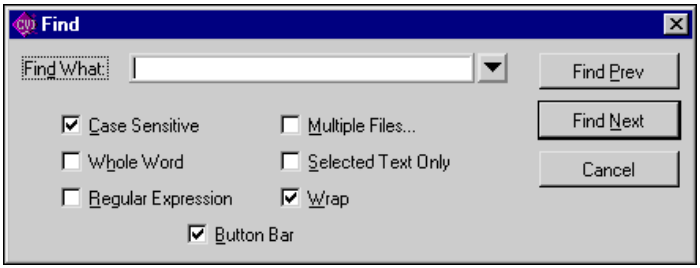


**Figure 4-8.** Find Dialog Box

Enter the text you want to find in the Find What box. If you select text on a single line before you execute the **Find** command, the selected text appears in the Find What box. Otherwise, the text you last searched for appears in the box. You can access a history of selections for the Find What box by clicking the mouse on the arrow to the right of the Find What box, or by using the up or down arrow keys.

The Case Sensitive option 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`.

The Whole Word option 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.

If you select the Regular Expression option, LabWindows/CVI treats certain characters in the Find What box as regular expression characters instead of literal characters. Table 4-1 describes the regular expression characters.

**Table 4-1.** 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`, etc. `a.*` matches `act`, `apt`, and `abort` |
| | `+` (plus sign) | Match 1 or more occurrences of preceding character or expression | `0+1` matches `01`, `001`, `0001`, . . . |
| Matching either/or | `?` (question mark) | Match 0 or 1 occurrences of preceding character or expression | `0?1` matches `1`, `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` |

**Table 4-1.** Regular Expression Characters (Continued)

| Purpose | Character | Description | Example |
|---|---|---|---|
| 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 |
| 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 '?' |

Use the Multiple Files option to include open source files and source files from the project in the search.

Use the Selected Text Only option to search only within the region of highlighted text 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.

Use Wrap to continue searching from the beginning of the file once the search has reached the end of the file.

Use the Button Bar option to enable or disable the built-in dialog box for interactive searching, as shown in Figure 4-9. **Find Prev** and **Find Next** operate the same as 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. Use the **Keyboard Help** command in the **Options** menu for a list of the search hot-keys.

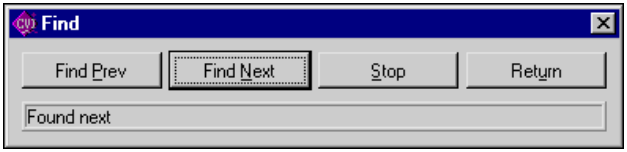


**Figure 4-9.** Find Button Bar

**Find Prev** searches for the closest previous occurrence of the specified text.

**Find Next** searches for the next occurrence of the specified text.

The **Stop** button cancels the **Find** command.

You can bypass the Find dialog box using the keyboard commands shown in Table 4-2.

**Table 4-2.** Keyboard Commands for Implementing Find

| Function | Key Combination |
|---|---|
| Find again (up) | <Ctrl-F3> |
| Find again (down) | <F3> |
| Use selected text as search string | <Ctrl-Shift-F3> |

## Replace. . .

The **Replace** command operates the same as the **Find** command except that you can replace the search string with another string. As LabWindows/CVI performs the search, a button bar appears, as shown in Figure 4-10.

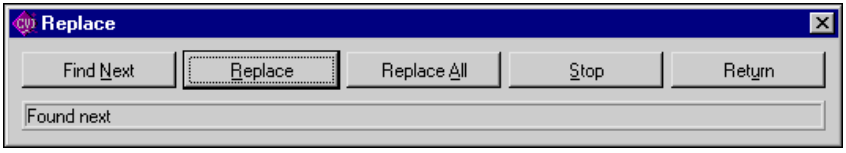


**Figure 4-10.** Replace Button Bar

**Find Next** skips to the next occurrence of the search string without making a change.

**Replace** executes the replacement.

**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.

You can bypass the Replace button bar using the keyboard commands shown in Table 4-3.

**Table 4-3.** Keyboard Commands for Implementing Replace

| Function | Key Combination |
|---|---|
| Find again (up) | <Ctrl-F3> |
| Find again (down) | <F3> |
| Use selected text as search string | <Ctrl-Shift-F3> |
| Replace selected text | <Ctrl-F11> |
| Replace selected text and find again | <F11> |
| Use selected text as replace string | <Ctrl-Shift-F11> |

## Next File

If you have selected the Multiple Files option from either the Find or Replace button bars, you can move to the next file in the search list using this command.

# View Menu

Use commands in the **View** menu for displaying line numbers and tags on source code, stepping through build errors, and manipulating function panels that pertain to your editing session.

Figure 4-11 shows the **View** menu.

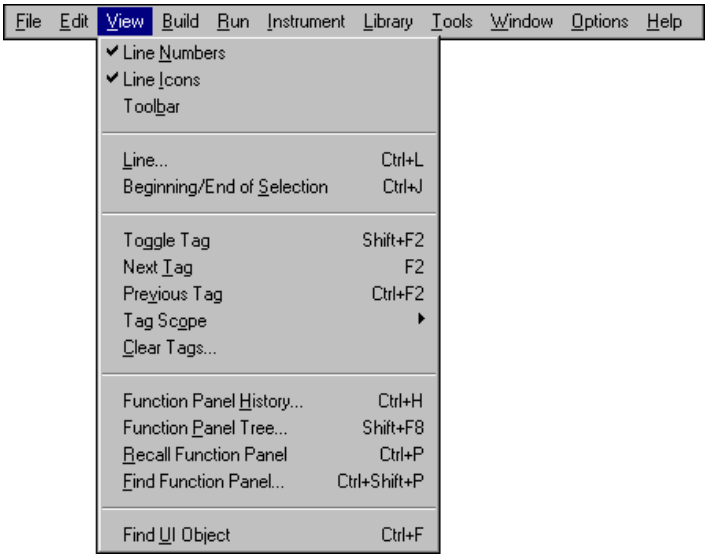


**Figure 4-11.** View Menu

## Line Numbers

The **Line Numbers** command controls the presence of line numbers in a window. A checkmark appears next to the **Line Numbers** item in the **View** menu when you activate the line number display.

## Line Icons

The **Line Icons** command controls the presence of line icons in a window. Line icons indicate the lines that you mark for breakpoint and the lines that you tag. A checkmark appears next to the **Line Icons** item in the **View** menu when line icons appear.

☞ **Note** ***LabWindows/CVI saves line icons in the project file. Editing source files outside of LabWindows/CVI, however, might invalidate the associated line icons.***

## Toolbar

Use the **Toolbar** command to toggle the visibility of the Source window toolbar.

## Line. . .

The **Line** command moves the cursor to the line that you specify. When you select the **Line** command, 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.

## 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.

## 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. Refer to the **Next Tag** and **Previous Tag** commands for more information.

## 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.

## 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 the **Tag Scope** command specifies.

## 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.

## Clear Tags. . .

Use the **Clear Tags** command to selectively remove existing tags.

## 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.

## Function Panel Tree

The **Function Panel Tree** command displays the Select Function Panel dialog box for the most recently used function panel.

## Recall Function Panel

When you are editing a function call in a Source window or the IEW, you might want to display the function panel corresponding to the call. You can do this with the **Recall Function Panel** command. The **Recall Function Panel** command not only finds and displays the panel, but also sets the panel controls so that they contain the parameter values appearing in the function call. After modifying one or more controls, you can replace the original call with the modified call.

### Invoking the Recall Function Panel Command

Before you invoke the **Recall Function Panel** command, you must indicate the function panel you want to recall. The simplest method is to place the cursor on a line that contains a function call or a portion of a function call. Also, you can select, or highlight, a range of lines that contains one or more function calls. Also, you can select part of a line, provided that 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 the **Recall Function Panel** command from the **View** menu. 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 simply type a function name into the Source window and execute **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. Refer to the *Find Function Panel. . .* section, which follows this section.

### 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 on 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 have to compile the file you are working in before you invoke the **Recall Function Panel** command. In fact, the function call you select does not have 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.

## Find Function Panel. . .

When you select the **Find Function Panel** command, 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 on **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use a regular expression as your search string. Refer to Table 4-1, *Regular Expression Characters*, for a list of regular expression characters.

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 shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

## Find UI Object

You use the **Find UI Object** command to move directly from a Source window to a User Interface Editor window. To use it, 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 the **Find UI Object** command from the **View** menu. 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 it finds an object, the User Interface window that contains the object comes to the foreground.

If the matching object is a panel, the panel's 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

Use the commands in the **Build** menu for compiling files, and building and linking projects. Figure 4-12 shows the **Build** menu.

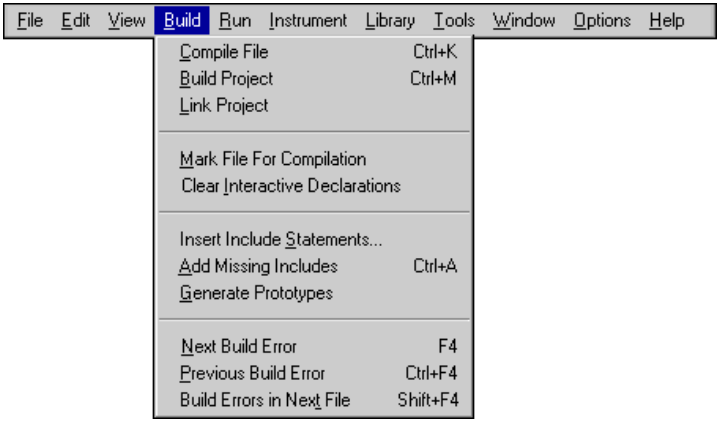


**Figure 4-12.** Build Menu

## Compile File

You must compile your source code before executing it in a Source window or the IEW. The **Compile File** command adds the file to the project if necessary, checks it for syntax errors, and compiles it. After LabWindows/CVI completes compilation, a Build Errors box appears if the file has any build errors.

When you want to call a function that you define in a Source window from another Source window, the IEW, 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.

**Note** ***The* Compile File *command is not available in the Standard Input/Output window.***

Refer to the *Compiler Options. . .* discussion in the *Options Menu* section of Chapter 3, *Project Window*, for a discussion of compiler options.

## Build Project

Use the **Build Project** command to compile all source files in the project that you or LabWindows/CVI marks for compilation and then to link the compiled files.

## Link Project

Use the **Link Project** command to link the compiled files from the project. **Link Project** does not invoke any compilation.

## Mark File for Compilation

When LabWindows/CVI marks a source file for compilation, a `C` appears next to the filename in the Project 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 manually force LabWindows/CVI to compile a source file on the next build by selecting the **Mark File for Compilation** command.

## Clear Interactive Declarations

Variables you declare in the IEW remain in effect until you explicitly remove them. This feature lets you use these variables in succeeding executions of the IEW. It also enables different function panels to access the same variables.

When you delete the entire contents of the IEW with the **Clear Window** command in the **Edit** menu, LabWindows/CVI removes the variables. If you want to remove the variables without deleting the contents of the IEW, use the **Clear Interactive Declarations** command.

## Insert Include Statements. . .

The **Insert Include Statements** command invokes a dialog box you can use to select one or more header files to include at the top of the program.

## 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 the missing prototypes. It 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** menu.

## 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.

## Next/Previous Build Error

If, when you compile a file or build your project, LabWindows/CVI displays multiple errors, you can use the **Next Build Error** command to step to the next build error. LabWindows/CVI highlights source code as you step through the errors. You can use the **Previous Build Error** command to step to the previous build error.

## 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

Use the commands in the **Run** menu to run and debug your program. Figure 4-13 shows the **Run** menu.

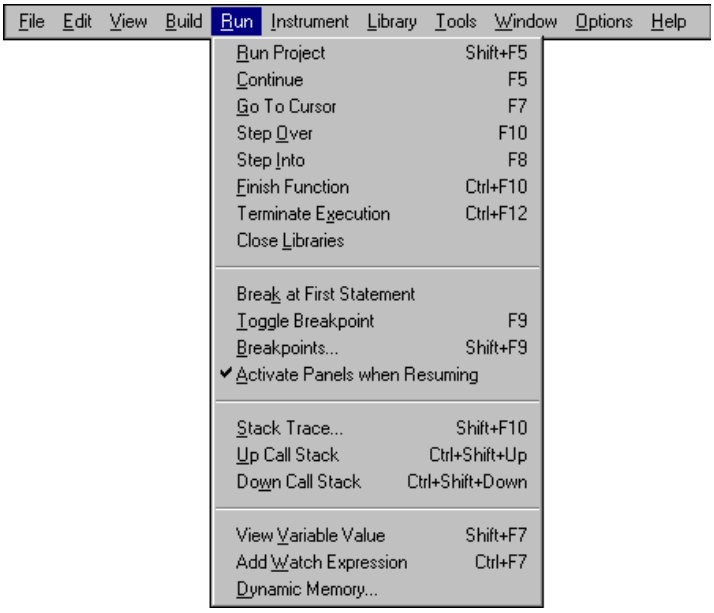


**Figure 4-13.** Run Menu

## Introduction to Breakpoints and Watch Expressions

It is important to understand the concepts of breakpoints and watch expressions before you learn about the commands in the **Run** menu.

You can pause the execution of a program without aborting it altogether by marking breakpoints in your 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 your source file. If you modify your source code outside of the LabWindows/CVI environment, you might invalidate breakpoint position information. You can use the `Breakpoint` function to specify a breakpoint directly in your 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.***

## 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 use many other features of the LabWindows/CVI environment. For instance, 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 IEW or a function panel.

To resume the execution after a breakpoint, you have several options under the **Run** menu that control the next breakpoint. You can restart the project at a breakpoint by selecting **Run Project**. To halt the execution of a program at a breakpoint, select **Terminate Execution** or press <Ctrl-Alt-SysRq> under Windows 3.1 or <Ctrl-F12> under Windows 95/NT and UNIX.

## Setting and Clearing Breakpoints

You can set and clear breakpoints in several ways.

- If you activate **Line Icons** in the **View** menu, click the mouse in 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 **Toggle Breakpoint** in the **Run** menu or press <Shift-F9>.
- Select **Breakpoints** from the **Run** menu to edit all breakpoints in the project and the IEW. You also can use the **Breakpoints** command to set *conditional* breakpoints. Refer to the *Conditional Breakpoints* section later in this chapter for information about conditional breakpoints.
- Select **Breakpoint at First Statement** from the **Run** menu to breakpoint on the first executable statement in the project or the IEW.
- You can set breakpoints directly in your source code using the `Breakpoint` function.
- You can manually suspend execution while your program runs, if your program checks for user input. For example, if your program makes calls to `RunUserInterface` or `scanf`, pressing <Ctrl-Alt-SysRq> under Windows 3.1 or <Ctrl-F12> under Windows 95/NT and UNIX causes a breakpoint state.

### Conditional Breakpoints

Set conditional breakpoints with the **Breakpoints** command in the **Run** menu. When you assign a conditional breakpoint to a line in your 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.

If you assigned these expressions to line 23 in your 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, however. Instead, LabWindows/CVI evaluates them before each statement in your source code. Refer to Chapter 6, *Variables and Watch Windows*, for more information about watch expressions.

## Run Project/Run Interactive Statements

### Running in a Source Window

The **Run Project** command compiles any source files in the project that you or LabWindows/CVI marks for compilation, links them together, and runs the project. Refer to the *Mark File for Compilation* discussion in the *Build Menu* section earlier in this chapter for information on marking files for compilation. If LabWindows/CVI encounters any build errors, it terminates the process and the Build Errors window appears with the list of errors.

### Running in the Interactive Execution Window

Select **Run Interactive Statements** in the IEW to execute code in that window. You do not have to enter a complete program in the IEW. For instance, you can execute variable declarations and assignment statements in C without declaring a `main` function.

You can use the IEW to test portions of code before including them in your main program. You also can use the IEW to execute functions exported by a loaded instrument or by a file in the project if the project has been linked. The IEW 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 IEW.

Refer to the *Interactive Execution Window* section for the rules governing code execution in the IEW.

LabWindows/CVI does not disturb asynchronous I/O, RS-232 ports, opened files, and User Interface Library resources you use in the IEW at the beginning or end of execution in the IEW. LabWindows/CVI terminates, closes, or deletes these program elements only when one of the following events occurs:

- You select **Clear Interactive Declarations** from the **Build** menu.
- You select **Clear Window** from the **Edit** menu.
- You link a project.
- You 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 the **Terminate Execution** command or the shortcut key <Ctrl-Alt-SysRq> under Windows 3.1 or <Ctrl-F12> under Windows 95/NT and UNIX.

# Continue

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

# 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 **Go To Cursor**. Program execution then continues until it reaches that line, where it enters another breakpoint state.

# Step Over

Use the **Step Over** command to execute an outlined statement when 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.

## 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.

## Finish Function

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

## Terminate Execution

The **Terminate Execution** command terminates a program that is suspended at a breakpoint. The shortcut key for terminating execution of a suspended program or suspending a running program is <Ctrl-F12> under Windows 95/NT and UNIX. Under Windows 3.1 the shortcut key is <Ctrl-Alt-SysRq>.

## Close Libraries

The **Close Libraries** command closes the LabWindows/CVI libraries you have accessed through function panels and the IEW. Use this command if you are working from function panels or the IEW and you want to close the LabWindows/CVI libraries without clearing your interactive variables.

**Note** ***LabWindows/CVI automatically closes the libraries before and after you run a project.***

## Break at First Statement

**Break at First Statement** is a run mode that enters a breakpoint state on the first executable statement in your source code. When activated, LabWindows/CVI puts a checkmark beside this command in the menu.

## Toggle Breakpoint

The **Toggle Breakpoint** command toggles the state of the breakpoint on the current line.

## Breakpoints. . .

The **Breakpoints** command opens the Breakpoints dialog box that contains a list of the breakpoints in the project, as illustrated in Figure 4-14. Also, you can open this dialog box by right-clicking in the line icons column.

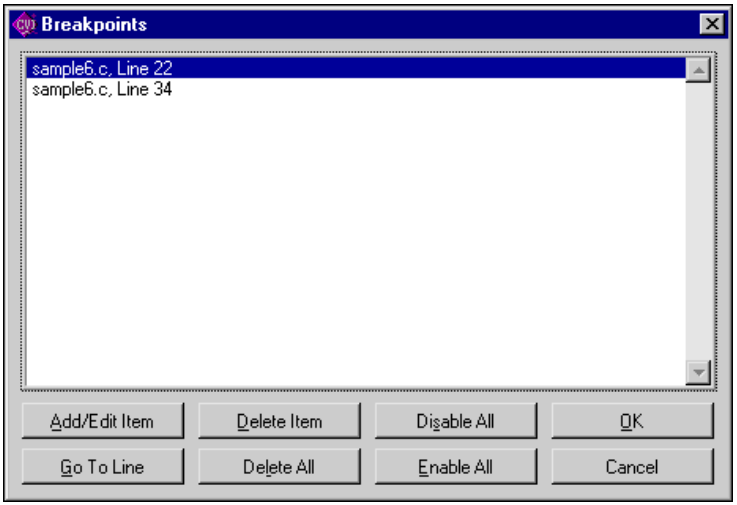


**Figure 4-14.** Breakpoints Dialog Box

Use the **Add/Edit Item** button to edit a single breakpoint with the Edit Breakpoint dialog box, as shown in Figure 4-15.

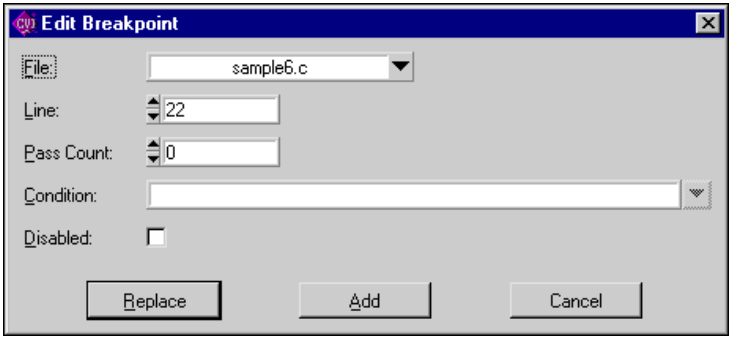


**Figure 4-15.** Edit Breakpoint Dialog Box

- **File**—Use this control to select the source file that contains the breakpoint you want to edit.
- **Line**—Use this control to select the line that contains the breakpoint you want to edit.

- **Pass Count**—Use this control to select the number of times that the source code line executes before the breakpoint occurs.
- **Condition**—Use this box to enter an optional expression that LabWindows/CVI evaluates before it executes the source code line. If the condition is true at that time, your program enters a breakpoint state; otherwise, execution continues. Refer to the *Conditional Breakpoints* section earlier in this chapter for examples of conditional expressions.
- **Disabled**—Use this checkbox to disable the breakpoint. The breakpoint icon in the Source window changes color to indicate that you disabled it.
- After you set all the breakpoint attributes in the dialog box, you can **Replace** the breakpoint with the new attributes, **Add** the breakpoint to the breakpoint list, or **Cancel** the operation.

The **Go to Line** button takes you to the source code location of the currently selected breakpoint.

The **Delete Item** button deletes the currently selected breakpoint.

The **Delete All** button deletes all the breakpoints.

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

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

The **OK** button accepts the current breakpoint attributes, and the **Cancel** button cancels the current operation.

## Activate Panels when Resuming

You can use **Activate Panels when Resuming** to choose whether the user interface panels in your programs reactivate every time you resume execution during debugging. By default, this option is enabled. Activating the panels whenever you resume guarantees that the activation state of every panel is identical to what it would be if you were not debugging. In general, however, this is not important, and activating panels each time you resume can be time consuming.

If you disable this option, LabWindows/CVI activates your panels when your program causes events to be processed or explicitly displays, activates, hides, or discards panels.

If you enable this option, LabWindows/CVI activates your most recently active panel upon resuming execution, but only if it was the active window at the time execution was suspended. If, for instance, you suspend execution by selecting the **Break Execution** command from the

**Run** menu in a Source window, none of your user interface panels activate when you resume execution.

LabWindows/CVI saves this option from one session to another, not in the project file.

## Stack Trace. . .

You can use the **Stack Trace** command only when in 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.

### Up Call Stack

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

### Down Call Stack

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

## 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 your selected variable highlighted.

## 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 select.

## Dynamic Memory. . .

The **Dynamic Memory** command opens a dialog box that displays the dynamic memory area. This area contains data that your program dynamically allocates using `malloc`, and any variables you declare in the IEW. You can view the memory in hexadecimal (byte, word, long), integer (byte, word, long), single-precision floating-point, double-precision floating-point, or ASCII representation.

# Instrument Menu

The **Instrument** menu for the Source, Interactive Execution, and Standard Input/Output windows works the same as the **Instrument** menu in the Project window. Refer to Chapter 3, *Project Window*, for information on the **Instrument** menu.

# Library Menu

The **Library** menu for the Source, Interactive Execution, and Standard Input/Output windows works the same as the **Library** menu in the Project window. Refer to Chapter 3, *Project Window*, for information on the **Library** menu.

# Tools Menu

This section explains how to use the commands in the Source window **Tools** menu, as shown in Figure 4-16.

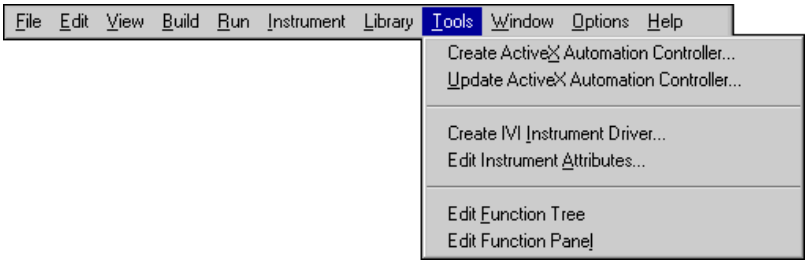


**Figure 4-16.** Tools Menu

## Create ActiveX Automation Controller. . . (Windows 95/NT Only)

Use the **Create ActiveX Automation Controller** command to generate a new instrument driver for an ActiveX Automation server. Refer to Chapter 3, *Project Window*, for more detailed information on the **Create ActiveX Automation Controller** command.

## Update ActiveX Automation Controller. . . (Windows 95/NT Only)

Use the **Update ActiveX Automation Controller** command to add new objects to an existing ActiveX Automation controller instrument driver. The command opens the Browse Type Library dialog box for the ActiveX Automation server the instrument driver controls.

In the dialog box, you can select additional server objects to add to the driver. The wizard adds the property constants, method functions, and creation functions for the objects to the driver.

If you select an object that was already included in the driver, the wizard gives you the option to replace the existing implementation or leave it as is.

## Create IVI Instrument Driver. . .

Use the **Create IVI Instrument Driver** command (the IVI wizard) 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:

- An existing driver for a similar instrument
- The core IVI driver template
- An IVI instrument class template

The IVI 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 on the IVI wizard.

## Edit Instrument Attributes. . .

Use the **Edit Instrument Attributes** command to add, delete, or edit attributes for an IVI instrument driver. You can invoke this command only if the file in the Source window has the same path and base file name 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. It 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 invoke the 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 on the **Edit Instrument Attributes** command.

## 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 name as the file in the Source window.

## 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 file name 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

The **Window** menu in the Source, Interactive Execution, and Standard Input/Output windows works the same as the **Window** menu in the Project window. Refer to Chapter 3, *Project Window*, for information on the **Window** menu.

# Options Menu

Use the commands in the **Options** menu to set up preferences in the LabWindows/CVI environment, and execute various LabWindows/CVI utilities.

Figure 4-17 shows the **Options** menu.

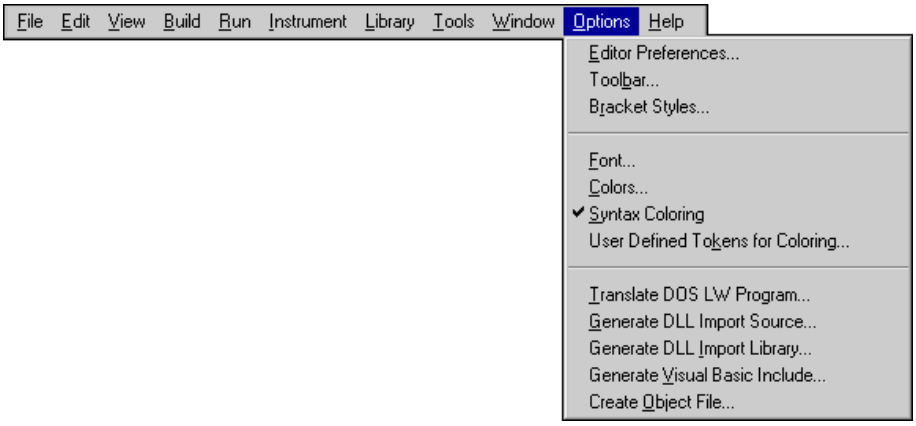


**Figure 4-17.** Options Menu

## Editor Preferences. . .

The **Editor Preferences** command invokes the dialog box shown in Figure 4-18, which you can use to set up Source window editor preferences.

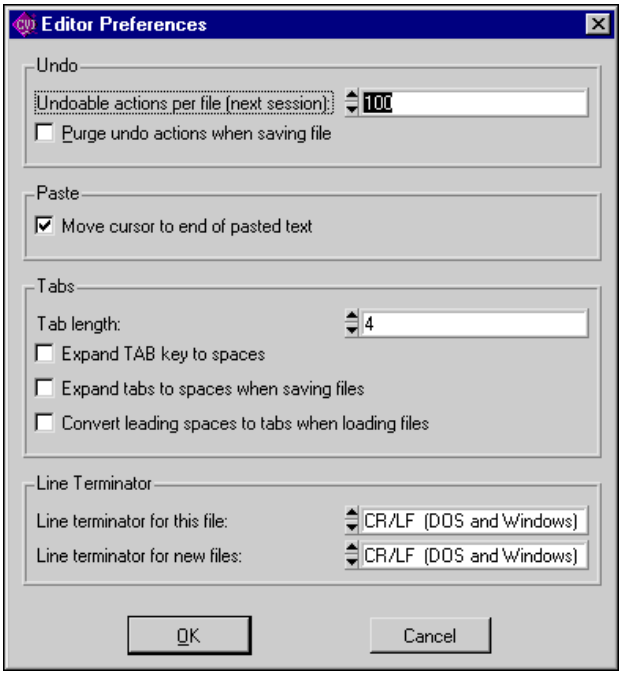


**Figure 4-18.** Editor Preferences

### Undo

Use the Undoable Actions per File option of the Editor Preferences dialog box to set the number of actions per file that you can undo. Select the Purge Undo Actions When Saving File option to clear the accumulated list of editing actions each time you save a file.

### Paste

Use the Paste section of the Editor Preferences dialog box to set where LabWindows/CVI places the text cursor after completing a paste operation. Select the Move Cursor to the End of Pasted Text option to put the cursor at the end of the pasted text. Leave this option unselected to put the cursor at the beginning of the pasted text.

### Tabs

Use the Tab Length option of the Editor Preferences dialog box to 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

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 your text file into DOS/Windows editors, select CR/LF termination.
- If you want to load your text file into a UNIX editor, select LF termination.
- If you want to load your text file into a Macintosh editor, select CR termination.

## Toolbar. . .

Use the **Toolbar** command to select which icons appear in the Source window toolbar.

## Bracket Styles. . .

The **Bracket Styles** command allows you to set the location of curly brackets when the following commands generate them in your program.

- The **Insert Construct** command in the **View** menu of the Source window.
- The **Generate** command in the **Code** menu of the User Interface Editor window.

You can specify two bracket styles: one for functions, and another for statements, such as `if` and `switch` statements.

## Font. . .

Use the **Font** command to select the font and font size for text in Source windows and Variables windows. You can select from a list of monospace fonts.

## Colors. . .

Refer to the *Options Menu* section of Chapter 3, *Project Window*, for a description of the **Colors** command.

## Syntax Coloring

When you enable the **Syntax Coloring** command, LabWindows/CVI color codes the various types of tokens in your source and include files. The following are the different types of tokens that can be color coded.

- C keywords
- Identifiers
- Comments
- Integers
- Real numbers
- Strings
- Preprocessor directives
- User-defined tokens

Set the color for a token type through the **Color** command in the **Options** menu.

Create the list of user-defined tokens through the **User Defined Tokens for Coloring** command in the **Options** menu.

## User Defined Tokens for Coloring...

Use the **User Defined Tokens for Coloring** command to 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 project file or from one LabWindows/CVI session to another.

## Translate DOS LW Program...

Use the **Translate DOS LW Program** command to convert a source file written in LabWindows for DOS so that it can run in LabWindows/CVI. Refer to Chapter 12, *Converting LabWindows for DOS Applications* in the *Getting Started with LabWindows/CVI* manual for details about converting `.c`, `.uir`, `.lbw`, and `.obj` files from LabWindows for DOS for use in LabWindows/CVI.

## Generate DLL Import Source... (Windows 95/NT Only)

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 do 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 **Static Library** from the submenu attached to the **Target** command in the **Build** menu of the Project 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.***

## Generate DLL Import Library. . . (Windows 95/NT Only)

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. A file dialog box then appears. Enter the pathname of the DLL.

The **Generate DLL Import Library** command generates a `.lib` file with the same base name as the include file. If you choose to create an import library for each compiler, LabWindows/CVI creates the files in subdirectories named `msvc`, `borland`, `watcom`, and `symantec`. LabWindows/CVI creates a copy of the library for the current compatible compiler in the directory of the DLL.

## Generate DLL Glue Source. . . (Windows 3.1 Only)

If you are using a DLL under Windows 3.1, LabWindows/CVI requires that *glue code* be present in a `.lib` or `.obj` file that accompanies the `.dll` file. The glue code is necessary for LabWindows/CVI, which is a 32-bit application, to access Windows 16-bit DLLs.

LabWindows/CVI automatically generates and compiles glue code when it loads the DLL based in the contents of the accompanying .h file. Sometimes the glue code LabWindows/CVI generates is sufficient. In other cases you must generate, modify, and compile the glue source code yourself.

Refer to the *16-Bit Windows DLLs* section of Chapter 2, *Using Loadable Compiled Modules*, in the *LabWindows/CVI Programmer Reference Manual* for details about generating DLL glue source code.

LabWindows/CVI enables the **Generate DLL Glue Source** command only in windows that contain include files. The include file must contain declarations for all user-callable functions on the DLL. The command generates glue source code into a new window.

## Generate DLL Glue Object. . . (Windows 3.1 Only)

If you do not want to modify the DLL glue code that LabWindows/CVI automatically generates, you can place your .dll file directly in the project. LabWindows/CVI generates the glue code when it loads the DLL. However, you can use the **Generate DLL Glue Object** command to create a compiled version of the DLL glue code. The DLL loads faster when you place the DLL glue object module, instead of the DLL, in the project.

LabWindows/CVI enables the **Generate DLL Glue Object** command only in windows that contain include files. The file must contain declarations for all user-callable functions in the .dll.

## Generate Visual Basic Include. . . (Windows Only)

This command generates a 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 Visual Basic.

## 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. They 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 the Compile into Object option in the Project window rather than the* Create Object File *command. You enable the Compile into Object option by double-clicking in the* O *icon column to the right of the source file name in the Project window. When you enable the* O *icon for a source file, LabWindows/CVI automatically compiles the source file into an object file whenever you build the project and you have modified the source file or its include files. The next time you open the project in LabWindows/CVI, you do not have to recompile the source files.***

For Windows 95/NT, the **Create Object File** command gives you the option of creating an object file for each of the compatible external compilers rather than just for the current compatible compiler. If you choose to create an object file for each compiler, LabWindows/CVI creates the files in subdirectories named `msvc`, `borland`, `watcom`, and `symantec`. LabWindows/CVI creates a copy of the object file for the current compatible compiler in the parent directory.

You can compile your file using a third-party compiler LabWindows/CVI supports. Refer to the *LabWindows/CVI Programmer Reference Manual* for more information on compatible external compilers. 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.

# Help Menu

The **Help** menu provides information about LabWindows/CVI. Figure 4-19 shows the **Help** menu.

**Figure 4-19.** Help Menu

The **Contents** command invokes the online help for LabWindows/CVI.

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

The **Search for Help On** command allows you to search for keywords used within the online help.

The **Visit LabWindows/CVI Web Page** command displays the LabWindows/CVI Web page if you have registered a Web browser on your system. This command is available only under Windows.

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

The **Keyboard Help** command invokes a scrollable list of keyboard shortcut keys. Appendix A, *Source Window Keyboard Commands*, shows these shortcut keys.

# 5

# Using Function Panels

This chapter describes how to use LabWindows/CVI function panels to generate code to call functions in any of the LabWindows/CVI libraries.

A function panel is an interface to the functions in the LabWindows/CVI libraries and instrument drivers. You can use function panels to help generate and test function calls within LabWindows/CVI.

A Function Panel window generates one or more function calls with function parameters you specify in the function panel. LabWindows/CVI can execute these functions immediately in the Interactive Execution window (IEW). When you execute a function panel, LabWindows/CVI copies the generated code to the IEW and executes it. The first time you execute a function panel for an instrument driver or library, LabWindows/CVI creates and executes an `#include` statement for the header file associated with the instrument driver or library. Other sections of this chapter discuss the relationship between a function panel and the IEW in more detail.

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 using the **Recall Function Panel** command in the **View** menu of the Source window.

Normally, you use function panels to call into instrument drivers that appear in the **Instrument** menu and libraries that appear in the **Library** menu. Refer to the *Using Instrument Drivers* section in Chapter 3, *Project Window*, for detailed information on the relationship between instrument drivers and function panels. Also, you can use function panels to call functions in the project, as long as the functions are declared in the IEW. You thus can create function panels for functions that you call frequently, even if you do not keep the functions in a separate file. Refer to the *LabWindows/CVI Instrument Driver Developers Guide* for detailed information about creating function panels.

## Accessing Function Panels

You can access a function panel for a library from the **Library** menu or for an instrument driver from the **Instrument** menu. After you select an instrument or library name, choose a panel by making selections from the Select Function Panel dialog box.

Functions are grouped in a multilevel structure called a *function tree*. This structure groups functions into various *classes* according to the operation they perform to make finding

individual functions easier. 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 Panel windows.

In certain cases, it is convenient to access library or instrument module function panels in a linear fashion, that is, by moving through the list of functions without using the tree structure. The Select Function Panel dialog box has a Flatten checkbox which replaces the function class hierarchy with a list of all function panels at or below the current level. Once you have selected a function panel, four function panel commands: **Previous Panel**, **Next Panel**, **First Panel**, and **Last Panel**, give you access to function panels in this linear manner. Refer to the *View Menu* section for more detailed information about using these commands.

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. Refer to the *View Menu* section later in this chapter, and the *View Menu* section in Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, for more information on using these commands.

Figure 5-1 shows a Function Panel window for an instrument driver function.

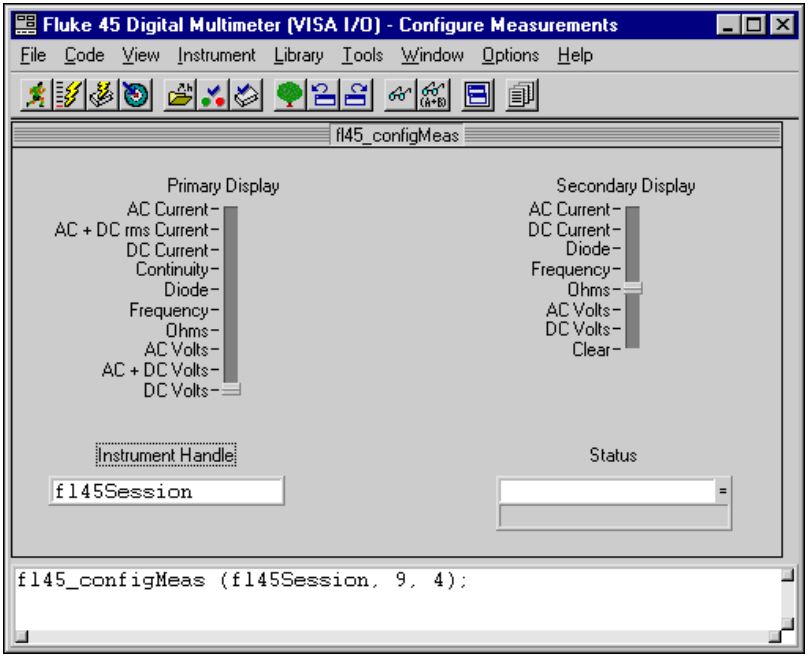


**Figure 5-1.** Instrument Driver Function Panel Window

Figure 5-1 shows the Configure Measurements Function Panel window for a Fluke 45 Digital Multimeter. It contains a function panel that corresponds to the `f145_configMeas` function. You can use the controls on the 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 **Function Call Disabled** from the **Options** menu. Disabled function calls do not appear in the Generated Code box, therefore, you cannot execute or insert them into a Source 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.

# Toolbars in LabWindows/CVI

The LabWindows/CVI toolbar appears within function panels, in the Function Panel Editor window, and in Source windows. It 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. Refer to the *Toolbars in LabWindows/CVI* section in Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, for a full description of toolbar use and configuration.

# Function Panel Controls

Function panel controls specify parameters in a function call. Figure 5-2 illustrates the eight types of function panel controls.

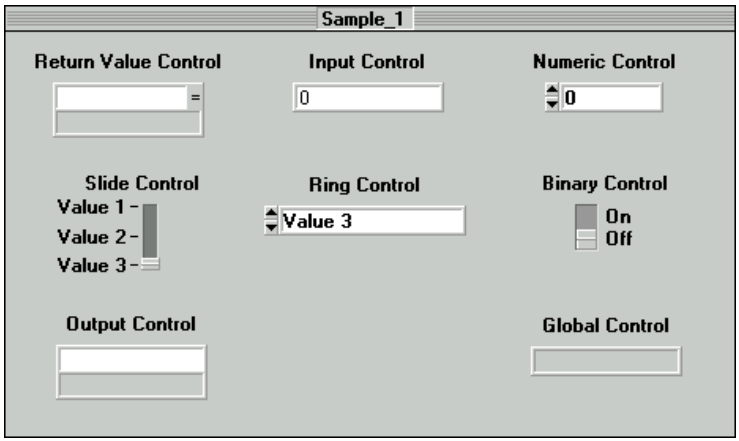


**Figure 5-2.** Function Panel Controls

When you open a function panel, input from the keyboard or mouse affects the currently selected control. Pressing the <Tab> key selects the next control. Pressing <Shift-Tab> selects the previous control. To select a control with the mouse, click on the control. Pressing <Page Up> or <Page Down> moves the input focus across multiple function panels in one window. Pressing <Ctrl-Page Up> and <Ctrl-Page Down> moves from one function panel window to the next.

The way you specify parameter values differs for each type of control. The following sections contain instructions for specifying parameters for each type of control.

## Specifying a Return Value Control Parameter

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 IEW or define it elsewhere and declare it as `extern` in the IEW before you execute the function. You can use the **Declare Variable** command from the **Code** menu to define the variables in the IEW. You can use the **Select Variable** command from the **Code** menu to select 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 window. After executing the function, the return value control displays the value for the variable beneath the variable name.

## Specifying an Input Control Parameter

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 in a variable name, numeric value, or valid expression. Before executing a function panel window, any names you type into input controls must be defined statically in the IEW or defined elsewhere and declared as `extern` in the IEW. You can use the **Declare Variable** command from the **Code** menu to define variables in the IEW for use in the function panels. You can use the **Select Variable** command from the **Code** menu 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 on the control to view the Help window.

## Specifying a Numeric Control Parameter

A *numeric control* behaves like an input control except that it accepts numeric values only.

If you want to type a variable name into a numeric control, use the **Toggle Control Style** command in the **Options** menu.

## Specifying a Slide Control Parameter

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.

To move the slider with the keyboard, press the up or down arrow key. As you move the slider, the corresponding argument in the function call in the Generated Code box changes. The <Home> and <End> keys move you to the top and bottom of the slide control, respectively. To move the slider with the mouse, click on the slider and drag it up and down, or just click on the position you want.

If you want to type a variable name into a slide control, use the **Toggle Control Style** command in the **Options** menu.

## Specifying a Binary Control Parameter

The *binary control* is a limited version of the slide control that has only two positions.

To select the position of the binary control, press the up or down arrow key, or the <Home> or <End> key. To change the binary control with the mouse, click on the position you want.

If you want to type a variable name into a binary control, use the **Toggle Control Style** command in the **Options** menu.

## Specifying a Ring Control Parameter

The *ring control* represents a range of values, much like the slide control. A ring control displays only a single item from a list, instead of displaying the whole list at once as the slide control does. The item you select determines the value LabWindows/CVI places in the function call.

To select an item from a ring control with the keyboard, use the up and down arrow keys to scroll through the list. Press the space bar to display the entire list of items for the selected ring control. To select an item from a ring control with the mouse, you can click on the up or down arrow of the ring control until the value you want appears, or you can click on the display field of the control and select the value you want directly from the list that appears.

If you want to type a variable name into a ring control, use the **Toggle Control Style** command in the **Options** menu.

## Specifying an Output Control Parameter

The *output control* displays a value that the function you execute determines.

To specify a parameter for an output control, select the control and type in the desired 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 IEW or defined elsewhere and declared as `extern` in the IEW before executing the function. You can use the **Declare Variable** command from the **Code** menu to define a variable in the IEW. You can use the **Select Variable** command from the **Code** menu to select a variable or expression that you have used before.

To view the value at an output control parameter after LabWindows/CVI executes the function, double-click on 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. These 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 *n* controls on a Common Control function panel specify the first *n* parameters of all functions in the Function Panel window.

## Convenient Viewing of Function Panel Variables

Select **View Variable Value** or **Add Watch Expression** from the **Code** menu for a convenient way 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

This section contains a detailed description of the **File** menu for Function Panel windows, as shown in Figure 5-3.

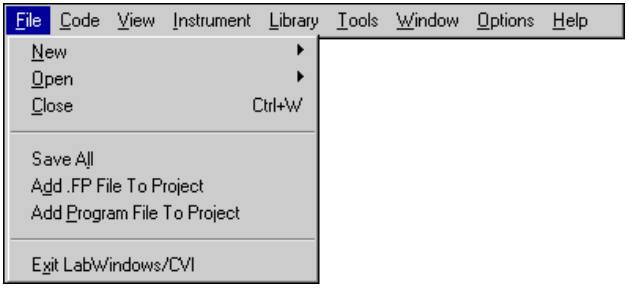


**Figure 5-3.** Function Panel Window File Menu

## New

The **New** command operates the same way as the **New** command in the Project window. Refer to the *File Menu* section in Chapter 3, *Project Window*, for more information on the **New** command.

## Open

The **Open** command operates the same way as the **Open** command in the Project window. Refer to the *File Menu* section in Chapter 3, *Project Window*, for more information on the **Open** command.

## Close

The **Close** command closes the active Function Panel window.

## Save All

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

## Add .FP File to Project

The **Add .FP File to Project** command adds the `.fp` file of the current Function Panel window to the project list.

## 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.

## Exit LabWindows/CVI

The **Exit LabWindows/CVI** command closes the current LabWindows/CVI session. If you have modified any open files since the last save, or if any windows contain unnamed files, LabWindows/CVI prompts you to save them to disk.

# Code Menu

This section contains a detailed description of the **Code** menu for Function Panel windows, as shown in Figure 5-4.

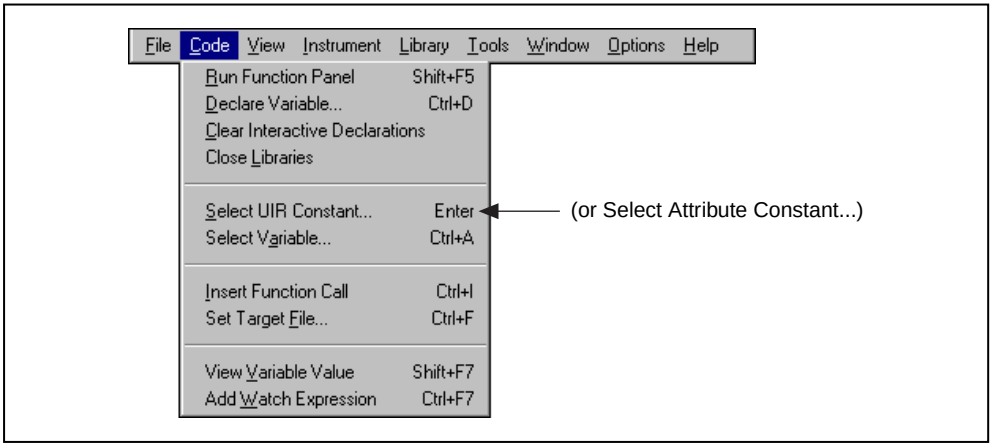


**Figure 5-4.** Code Menu

## Run Function Panel

Selecting the **Run Function Panel** command executes 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. While executing, 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.

## Declare Variable. . .

Use **Declare Variable** to declare a variable to be placed in the currently active control on the function panel. When you select **Declare Variable**, a dialog box appears, as shown in Figure 5-5.

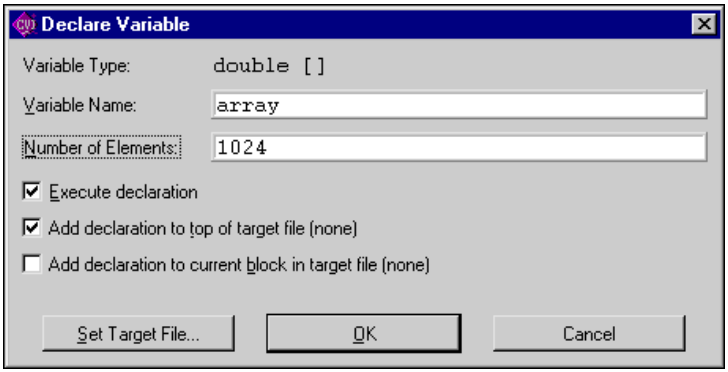


**Figure 5-5.** Declare Variable Dialog Box

To declare a variable with the Declare Variable dialog box, enter the name of the variable you want to declare in the Variable Name text box. LabWindows/CVI automatically prefixes scalar output variables with an ampersand (&).

The Variable Type message 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 selector allows you to select the data type.

The Number of Elements box appears when the currently active control is for an array or a string. Enter the number of elements.

Select the action options you want. When you select the Execute Declaration option, LabWindows/CVI executes the variable declaration immediately in the IEW. When you select the Add Declaration to Top of Target File (*filename*) option, LabWindows/CVI inserts a copy of the declaration at the top of the file you selected using the **Set Target File** button. When you select the Add Declaration to Current Block in Target File (*filename*) option, LabWindows/CVI inserts a copy of the declaration at the beginning of the code block that contains your current position.

Click on the **OK** button to declare the variable according to the options you have selected.

Click on the **Cancel** button to cancel the operation and remove the Declare Variable dialog box from the screen.

When you use the **Declare Variable** command, LabWindows/CVI always declares the variable using the static storage class unless you select the Add Declaration to Current Block in Target File option.

In addition to generating the variable declaration, the **Declare Variable** command 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.

## Clear Interactive Declarations

Variables declared in the IEW remain in effect until you explicitly remove them. Because of this feature, you can use these same variables in succeeding executions of the IEW, and different function panels can access the same variables.

The **Clear Interactive Declarations** command removes the variables without deleting the contents of the Interactive Execution window.

## Close Libraries

The **Close Libraries** command closes the LabWindows/CVI libraries you have accessed through function panels and the IEW. This command is useful if you are working from function panels or the IEW and you want to close the LabWindows/CVI libraries without clearing your interactive variables.

**Note** ***LabWindows/CVI automatically closes the libraries before and after you run a project.***

## Select UIR Constant. . .

The **Select UIR Constant** command can help you use the function panels for the User Interface Library. The command lets you select from the list of constant names associated with the objects in your `.uir` files.

## Selecting Constants from .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, as shown in Figure 5-6.

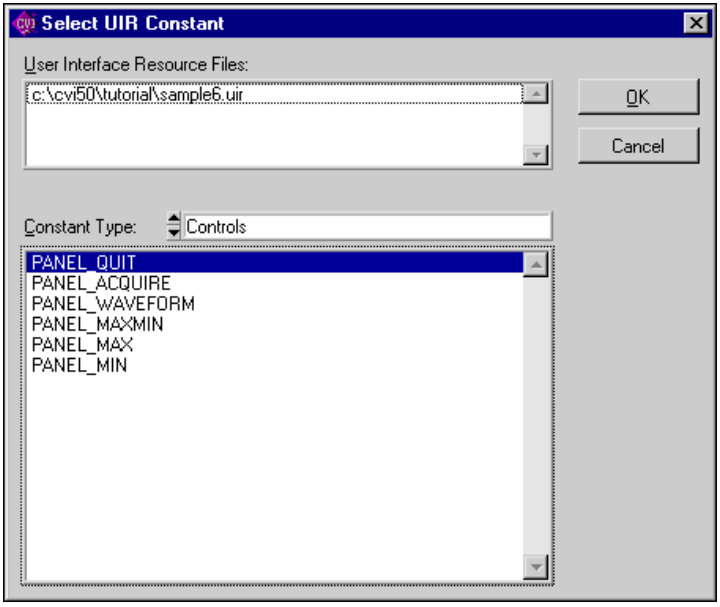


**Figure 5-6.** Select UIR Constant Dialog Box

The list box at the top of the dialog box lists all the `.uir` files open or in the project. Only constants from the currently selected `.uir` file appear in the list box at the bottom. Click on a file to select it.

The Constant Type ring control allows you to select which category of constant name to show.

When you click on the **OK** button, LabWindows/CVI copies the currently selected constant name into the function panel control.

☞ **Note** ***If you attempt to use* Select UIR Constant *on the Panel Handle and Menu Bar Handle controls that appear on most User Interface Library function panels, an error message appears. These controls take the values returned from*** `LoadPanel` ***and*** `LoadMenuBar`***, so an attempt to select*** `.uir` ***constants will fail.***

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.

## Select Attribute Constant. . .

In certain cases, the **Select Attribute Constant** command replaces the **Select UIR Constant** command in the **Code** menu. This occurs in panels for functions 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.

## Selecting Constants in an Attribute Control

When you execute the command on an Attribute ring control, the Select Attribute Constant dialog box appears, as shown in Figure 5-7.

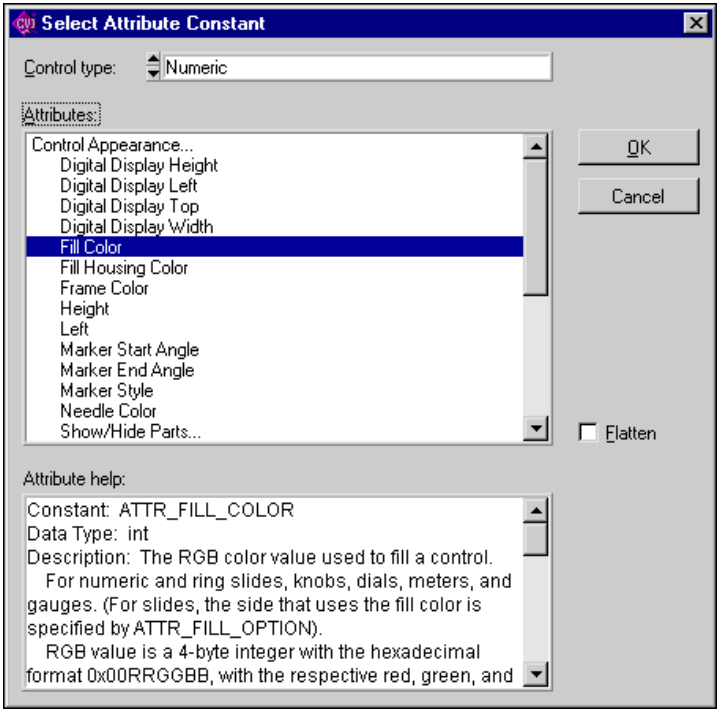


**Figure 5-7.** Select Attribute Constant Dialog Box

The Attributes list box displays the attributes you can use with the function. The attributes are organized under classes. A trailing ellipsis (. . .) denotes a class. To see a list of all the attributes without the classes and in alphabetical order, select the Flatten option. The name of

the attribute is dim if it does not allow the type of access that the function performs. For example, read only attributes for user interface controls appear dim when you use this dialog box from the function panel for `SetCtrlAttribute`.

The Control Type ring appears only when you use this dialog box from a User Interface Library function panel. It allows you to restrict the list of attributes to those applicable to a particular control type.

The Data Type ring appears when you use this dialog box from a function panel for a typesafe attribute function. The IVI Library and IVI instrument drivers have typesafe attribute functions, such as `Ivi_SetAttributeViInt32`. The Data Type ring allows you to restrict the list of attributes to those that have the same data type as the typesafe function. If you choose the see all attributes, the data type of each attribute appears on the right-hand side of the list box. The data type is dim if it is not the same as the data type of the typesafe function.

The Attribute help text box displays help information for the currently selected attribute.

Double-click on an attribute or click on **OK** to change the function panel ring control to that attribute. If you select an attribute that appears dim in the list box, an error message appears informing you that the attribute does not allow the type of access the function requires. If you select an attribute for which the data type appears dim in the list box, a dialog box appears giving you the option to change to the function panel for the typesafe function that you can use with that attribute.

Notice that when you attempt to operate the Attribute ring control in the function panel as a normal ring control, the same dialog box appears in place of the pop-up menu that normally appears on a ring control.

## Selecting Constants in a Value Control

The **Select UIR Constant** command has special behavior on the Attribute Value input and output controls in panels for functions such as `GetCtrlAttribute`, `SetCtrlAttribute`, `GetPanelAttribute`, and `SetPanelAttribute`.

When you execute the **Select Attribute Constant** command on an Attribute Value control, the behavior depends on the attribute currently selected in the Attribute ring control on the same function panel.

If you set the Attribute ring control to an attribute for which there is no small, discrete set of values, a dialog box appears repeating the help information for the attribute. If, on the other hand, there is a small, discrete set of values, the Select Attribute Value dialog box appears, as in Figure 5-8.

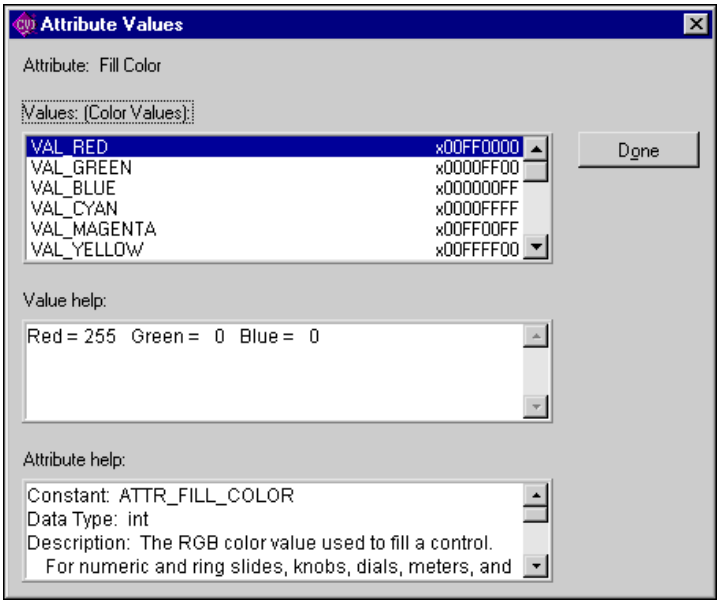


**Figure 5-8.** Select Attribute Value Dialog Box

When the values that appear in the Values list box are constant names, the actual values appear on the right-hand side of the list box.

If the Attribute Value control is an input control, such as on `SetCtrlAttribute` or `SetPanelAttribute`, double-click on an entry in the Values list to copy it into the Attribute Value control on the function panel.

If the Attribute Value control is an output control, such as on `GetCtrlAttribute` or `GetPanelAttribute`, and a value appears in the bottom half of the control because you executed the function panel, LabWindows/CVI selects, when possible, the value in the Values list that corresponds to the value shown in the bottom half of the output control. An arrow symbol appears to the left of the list box entry that contains that value.

## Select Variable. . .

The **Select Variable** command gives you 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.

When you execute the **Select Variable** command, the Select Variable or Expression dialog box appears, as shown in Figure 5-9.

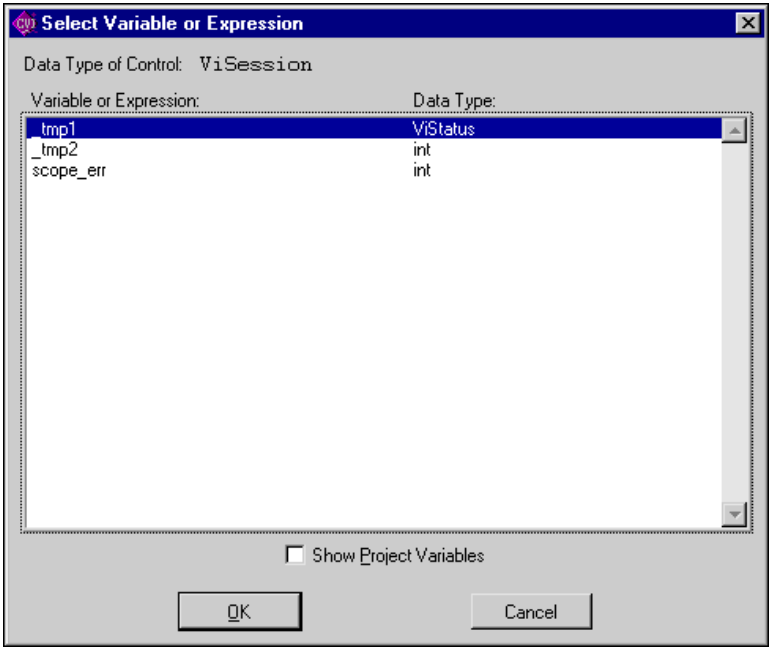


**Figure 5-9.** Select Variable or Expression Dialog Box

- **Data Type of Control**—Indicates the data type of the currently active function panel control.
- **Variable or Expression**—This list box column contains the variables and expressions that have data types compatible with the data type of the control.
- **Data Type**—This list box column indicates the data type of each variable and expression.
- **Show Project Variables**—This option adds to the list box global variables, static and non-static, defined in project files that have been successfully compiled.
- **OK**—This button dismisses the dialog box and copies the variable or expression into the function panel control. It might add a leading ampersand (`&`) when the function panel control is an output control. It might add one or more leading asterisks (`*`) or a trailing array indexation (`[0]`) when necessary to correctly match the data type of the control.
- **Cancel**—This button cancels the operation.

## What Is Included in the List Box

**Select Variable** considers the following items for inclusion in the list box:

- Variables you declare in the IEW.
- 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 the 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, but only if the Show Project Variables option is enabled in the dialog box.

LabWindows/CVI removes some or all these items from memory when you unload the current project or execute the **Clear Interactive Declarations** command in the **Build** menu.

## 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 IEW 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;
  ```

  Then 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 are all compatible:

  ```
  int
  int *
  ```

```
int **
int []
```

To be included in the list box, an expression or a variable name 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 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.***

### Sorting of 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.

## 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 IEW. You determine the destination window with the **Set Target File** command in the **Code** menu.

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. Refer to the discussion of the **Recall Function Panel** command in the *View Menu* section of Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, for more information.

## Set Target File. . .

Use the **Set Target File** command to set the destination file for the **Insert Function Call** command. **Set Target File** brings up a dialog box from which you can select from any open Source window or the IEW.

## View Variable Value

The **View Variable Value** command is a convenient way 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.

## 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 **Add Watch Expression**. The Watch window appears with the expression highlighted.

# View Menu

This section contains a detailed description of the **View** menu for Function Panel windows, as shown in Figure 5-10.

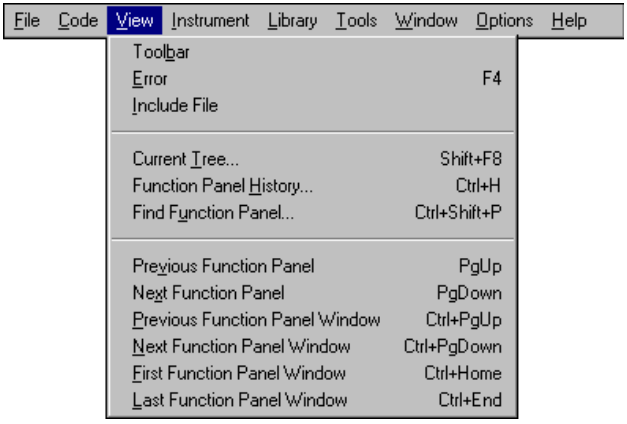


**Figure 5-10.** View Menu

## Toolbar

Use the **Toolbar** command to toggle the visibility in the Function Panel window toolbar.

## 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.

## 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.

## 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.

## 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.

## Find Function Panel. . .

When you select the **Find Function Panel** command, a dialog box appears in which you can enter the name of a function. You can enter just a substring, and the **Find Function Panel** command finds all functions that contain that substring anywhere in their names. For instance, if you enter `ctrl` and click on **OK**, a dialog box appears with a list of functions including `NewCtrl`, `SetCtrlVal`, `GetCtrlVal`, and so on.

You can use a regular expression as your search string. Refer to Table 4-1, *Regular Expression Characters*, for a list of regular expression characters.

If a function panel exists for the function, LabWindows/CVI displays the panel. If two or more function panels exist for the function, LabWindows/CVI displays a list of the function panels.

The shortcut key for **Find Function Panel** is <Ctrl-Shift-P>.

## Previous Function Panel

The **Previous Function Panel** command displays the previous function panel in the current Function Panel window.

## Next Function Panel Window

The **Next Function Panel** command displays the next function panel in the current Function Panel window.

### 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 Tree.

### 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.

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.

### First Function Panel Window

The **First Function Panel Window** command displays the first Function Panel window in the Function Tree.

### Last Function Panel Window

The **Last Function Panel Window** command displays the last Function Panel window in the Function Tree.

## Instrument Menu

The **Instrument** menu in Function Panel windows behaves the same way as the **Instrument** menu in the Project window. Refer to the *Instrument Menu* and *Using Instrument Drivers* sections in Chapter 3, *Project Window*, for command descriptions.

## Library Menu

The **Library** menu for Function Panel windows behaves the same way as the **Library** menu in the Project window. Refer to the *Library Menu* section in Chapter 3, *Project Window*, for command descriptions.

## Window Menu

The **Window** menu in Function Panel windows behaves the same way as the **Window** menu in the Project window. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for command descriptions.

# Options Menu

This section contains a detailed description of the **Options** menu for Function Panel windows, as shown in Figure 5-11.

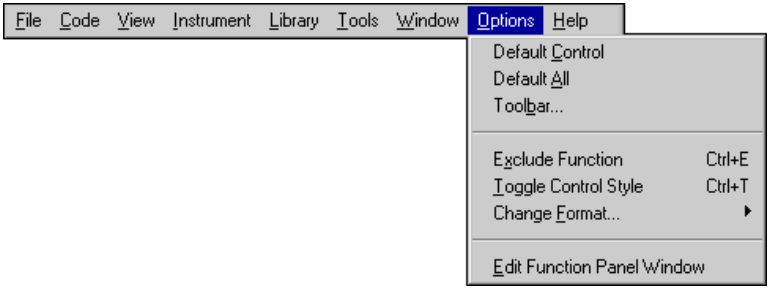


**Figure 5-11.** Options Menu

## Default Control

**Default Control** resets a control to its default value and configuration.

## Default All

**Default All** resets all the controls on the current Function Panel window to their default values and configurations.

## Toolbar. . .

Use the **Toolbar** command to select which icons appear in the Function Panel window toolbar.

## 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 use the **Run** or **Insert** commands from the **Code** menu.

## 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.

## Change Format. . .

**Change Format** lets you 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.

## Edit Function Panel Window

The **Edit Function Panel Window** command puts the Function Panel window in edit mode. Refer to Chapter 3, *The Function Panel Editor*, in the *LabWindows/CVI Instrument Driver Developers Guide* for information about editing instrument function panels.

☞ **Note** ***You cannot edit the function panels of the LabWindows/CVI libraries or user libraries.***

# Help Menu

The **Help** menu provides information about the current function panel and its controls. The **Help** menu is shown in Figure 5-12.

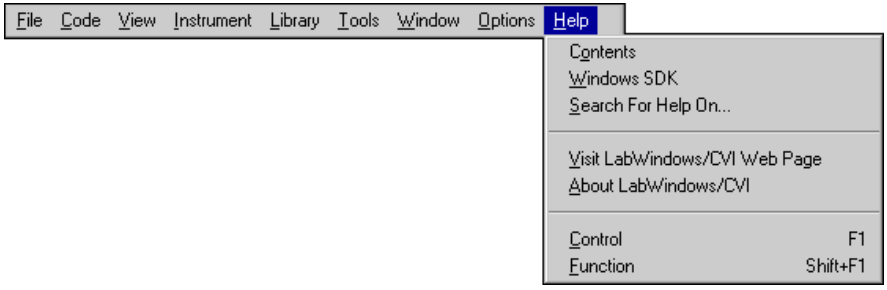


**Figure 5-12.** Help Menu

## Control

The **Control** command displays help information about the currently highlighted control. To select control help with the mouse, right-click anywhere on the control you want help with.

## Function

The **Function** command displays general information about the function the current Function Panel generates. To select function help with the mouse, right-click on the Function Panel.

# 6

# Variables and Watch Windows

This chapter describes the Variables and Watch windows. You use these windows to inspect and modify the values of program variables.

You can invoke these windows when no program is running or when a program is suspended at a breakpoint.

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). You can see the difference by displaying the string in decimal format.***

## Variables Window

To view the Variables window, select **Window»Variables** in the active LabWindows/CVI window. You also can invoke the Variables window for the currently highlighted variable from a Source or Function Panel window with the **Run»View Variable Value** command in the

Source window, or the **Code»View Variable Value** command in the Function Panel window. An example of the Variables window appears in Figure 6-1.

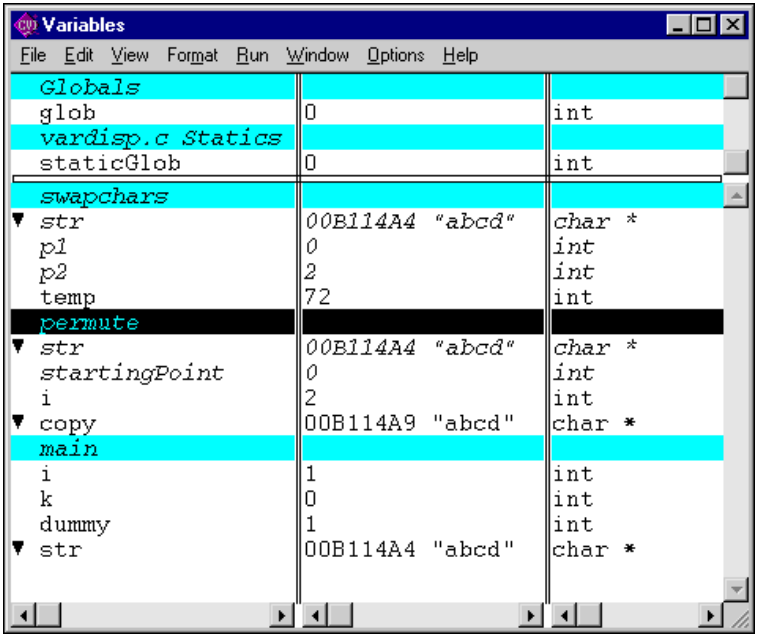


**Figure 6-1.** Variables Window

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 is also divided into two horizontal sections; the Global subwindow and the Function subwindow.

The Global subwindow displays the following variables:

- Project globals which include 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 appears in a different section. For any given function, the Variables window lists formal parameters first, followed by local variables. Formal parameters appear in italics.

Several icons appear to the left of certain variables as shown below:

- The variable on this line is the starting pointer to a block of defined data such as an array, string, or structure. Click on this icon or select **Expand Variable** from the **View** menu to expand the variable so that you can see each element or member. Refer to the *Expand Variable* discussion in the *View Menu* section later in this chapter.
- The variable on this line is the starting pointer to a block of defined data that appears in expanded form. Click on this icon or select **Close Variable** from the **View** menu to close the variable so that you see only the starting pointer. Refer to the *Close Variable* discussion in the *View Menu* section later in this chapter.
- The variable on this line is a member of a structure which is a parent pointer to another structure of the same type. Click on this icon or select **Follow Pointer Chain** from the **View** menu to replace the current structure with the child structure that the pointer references. Refer to the *Follow Pointer Chain* discussion in the *View Menu* section later in this chapter.
- The variable on this line is a child structure in a chain; the pointer to its parent structure does not appear. Click on this icon or select **Retrace Pointer Chain** from the **View** menu to replace the current structure with its parent. Refer to the *Retrace Pointer Chain* discussion in the *View Menu* section later in this chapter.

# 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 activate the Watch window, select **Window»Watch** in the active LabWindows/CVI window. Figure 6-2 shows an example of the Watch window.

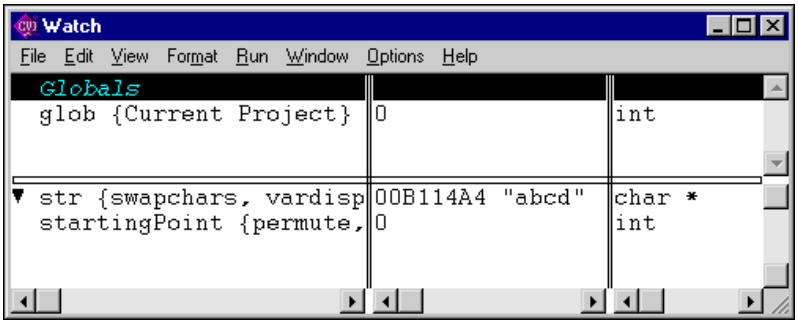


**Figure 6-2.** Watch Window

Select Watch window variables from the Variables window using the **Options»Add Watch Expression** command. The **Watch** command opens the dialog box shown in Figure 6-3.

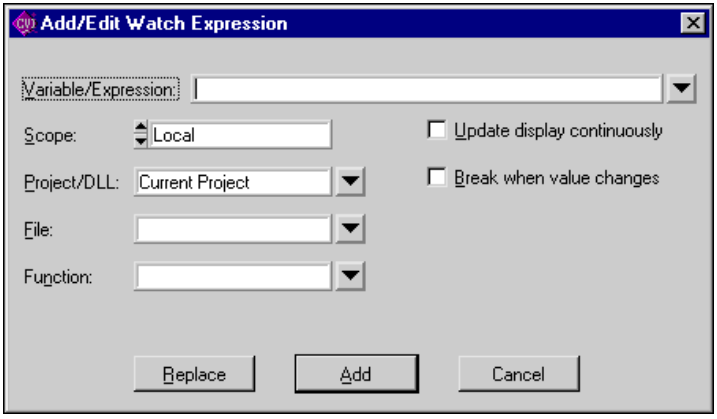


**Figure 6-3.** Add/Edit Watch Expression Dialog Box

Use the controls in the Add/Edit Watch Expression dialog box as follows:

- **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.
- **Project/DLL**—Indicates whether the watch expression applies to the current project or to a separate debuggable DLL. The default value for the control is Current Project. If you want the watch expression to apply to a DLL that is not the target of the current project, you must supply the name of the DLL. Enter the filename and extension, without a directory path, as in `mydll.dll`. The menu ring to the right of the control contains the names of all DLLs currently loaded. 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**—Name of the file that defines the variable or expression variables, if they are global to a file or local to a function.
- **Function**—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 on 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 only available 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 window. 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 window, highlight the expression and select **Code»Add Watch Expression**.

# File Menu

This section contains a detailed description of the **File** menu for the Variables and Watch windows.

Figure 6-4 shows the **File** menu.

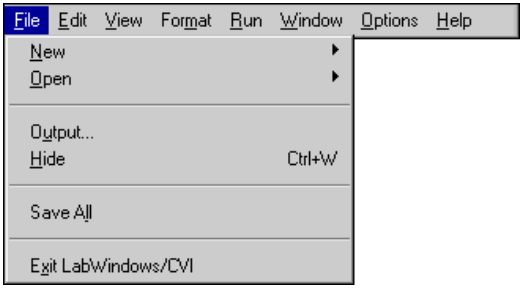


**Figure 6-4.** File Menu

### New

The **New** command operates the same way as in the Project window. For a description of this command, refer to the discussion of the **File»New** command in Chapter 3, *Project Window*.

### Open

The **Open** command operates the same way as in the Project window. For a description of this command, refer to the discussion of the **File»Open** command in Chapter 3, *Project Window*.

### 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.

### Hide

The **Hide** command visually closes a window while retaining the contents in memory.

### Save All

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

### Exit LabWindows/CVI

The **Exit LabWindows/CVI** command closes the current LabWindows/CVI session. If you have modified any open files since the last save, or if any windows contain unnamed files, LabWindows/CVI prompts you to save them to disk.

## Edit Menu for the Variables Window

This section contains a detailed description of the **Edit** menu for the Variables window.

Figure 6-5 shows the **Edit** menu for the Variables window.

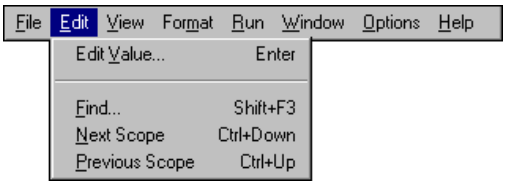


**Figure 6-5.** Edit Menu in the Variables Window

## Edit Value. . .

You can change the value of a variable with the **Edit Value** command. You can invoke the **Edit Value** command with the mouse by double-clicking on the variable name. When the dialog box appears, type in the new value.

The value that you enter in the Edit 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 reals in either scientific or floating-point format, regardless of the display format.
- Edit individual array elements by expanding the array using the **Expand Variable** command.
- Edit individual bytes of a string by expanding the string using the **Expand Variable** command. The bytes appear in the integer format you specify in the **Format** menu.

## Find. . .

The **Find** command invokes the dialog box shown in Figure 6-6.

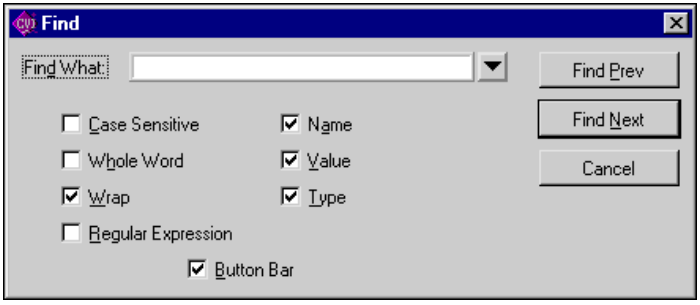


**Figure 6-6.** Find Dialog Box in the Variables Window

- **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. The characters `A` through `Z`, `a` through `z`, `0` through `9`, and underscore (_) are 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 the Find What box as regular expression characters instead of literal characters. Table 4-1 of Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, describes the regular expression characters.

- **Name**—Activate this option to include the variable name field of the Variables/Watch window in the search.
- **Value**—Activate this option to include the value field of the Variables/Watch window in the search.
- **Type**—Activate this option to include the variable type field of the Variables/Watch window in the search.
- **Button Bar**—Use this option to enable or disable the built-in dialog box for interactive searching, as shown in Figure 6-7.

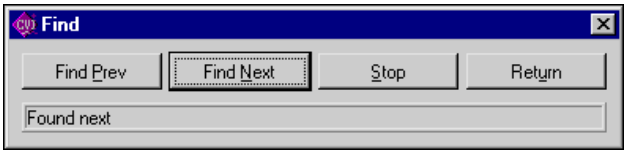


**Figure 6-7.** Find Button Bar

**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 hot keys remain active even if you disable the Button Bar. The search hot-keys in the Variables and Watch windows are the same as the search hot-keys in Source windows. Use the **Keyboard Help** command in the **Options** menu of a Source window for a list of the search hot-keys or refer to Appendix A, *Source Window Keyboard Commands*.

## 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 is not available in the Watch window.

## 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 is not available in the Watch window.

# Edit Menu for the Watch Window

Figure 6-8 shows the **Edit** menu for the Watch window.

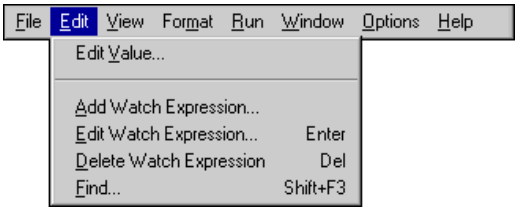


**Figure 6-8.** Edit Menu in the Watch Window

## Edit Value. . .

The **Edit Value** command operates the same way as in the Variables window.

## Add Watch Expression. . .

**Add Watch Expression** invokes the Add/Edit Watch Expression dialog box. The *Watch Window* section earlier in this chapter explains this dialog box.

## Edit Watch Expression. . .

**Edit Watch Expression** invokes the Add/Edit Watch Expression dialog box for the selected watch expression.

## 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.

## Find. . .

The **Find** command operates the same as in the Variables window.

# View Menu

This section contains a detailed description of the **View** menu for the Variables and Watch windows.

Figure 6-9 shows the **View** menu.

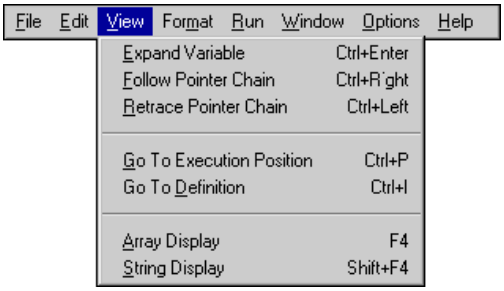


**Figure 6-9.** View Menu

To use one of these commands, select a particular array or string by clicking on it with the mouse or using the up and down arrow keys, and then access the command from the **View** menu.

## 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, as shown in Figure 6-10.

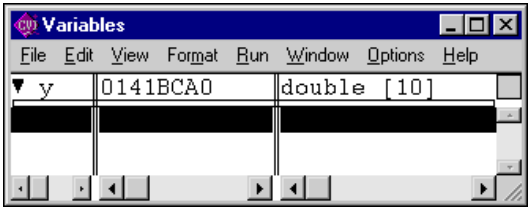


**Figure 6-10.** A Closed Array in the Variables Window

In expanded form, the icon changes to a circle and you see the individual elements and their values, as shown in Figure 6-11.

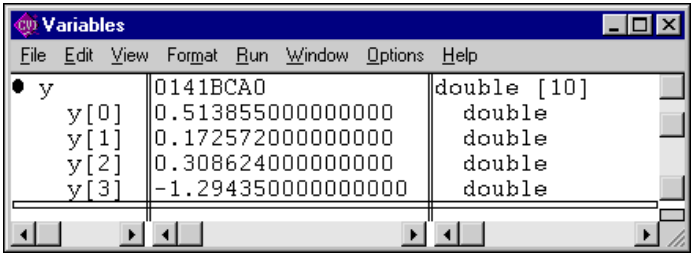


**Figure 6-11.** An Expanded Array in the Variables Window

The **Expand Variable** command expands a currently closed aggregate variable so you can see its contents. Clicking on the triangle icon has the same effect as selecting **Expand Variable**.

## Close Variable

Refer to the *Expand Variable* section for a discussion of expanded and closed variables.

The **Close Variable** command closes the currently expanded aggregate variable so you can see its name and starting address. Clicking on the circle icon has the same effect as selecting **Close Variable**.

## 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 6-12, `hquework->begin->next` is a member of the structure `hquework->begin` and points to another structure type of `Item`.

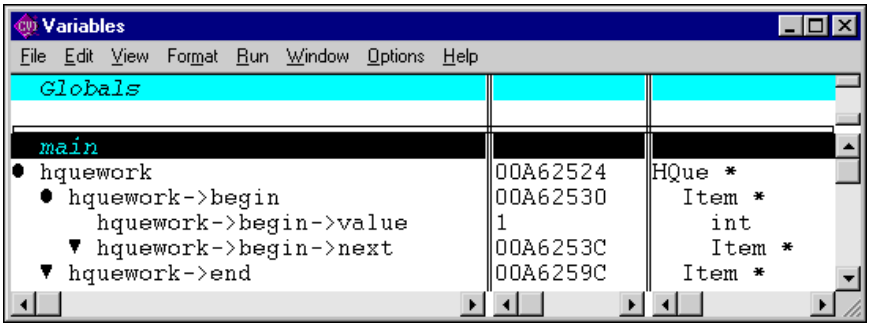


**Figure 6-12.** A Parent Structure Pointer in a Chain

Clicking on the right arrow icon or selecting **Follow Pointer Chain** replaces the current structure with the child structure that the pointer references, as shown in Figure 6-13.

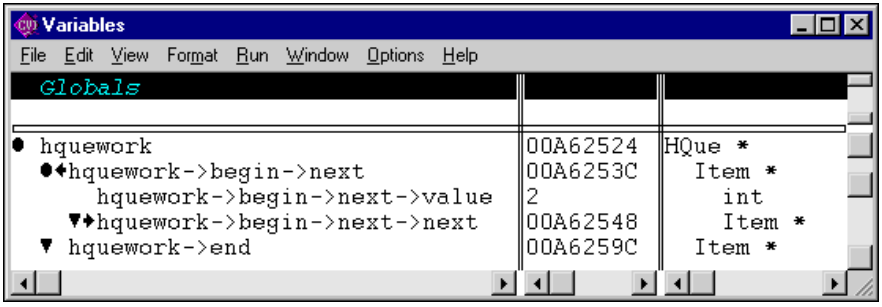


**Figure 6-13.** A Child Structure Pointer in a Chain

## Retrace Pointer Chain

**Retrace Pointer Chain** replaces the current structure with its parent. Notice the presence of the left arrow icon in Figure 6-13 after selecting **Follow Pointer Chain**. This indicates that the structure `hquework->begin->next` is a child structure in a chain. Clicking on the left arrow icon or selecting **Retrace Pointer Chain** causes the variable display to revert back to Figure 6-12.

☞ **Note** **Retrace Pointer Chain** ***is valid only when you displayed the current structure with*** **Follow Pointer Chain.**

## Go To Execution Position

The **Go To Execution Position** command is available only 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 on the function name or press <Ctrl-P>. This command is valid only in the Variables window.

## Go To Definition

The **Go To Definition** command opens the Source window that contains the definition of the currently selected function or variable, and highlights the definition. This command is valid only in the Variables window.

## Array Display

The **Array Display** command invokes the Array Display window for the currently highlighted array. To invoke the Array Display window, you can double-click on an array variable or press <F4>. Refer to Chapter 7, *Array and String Display Windows*, for more information.

## String Display

The **String Display** command invokes the String Display window for the currently highlighted string. To invoke the String Display window, you can double-click on a string variable or press <Shift-F4>. Refer to Chapter 7, *Array and String Display Windows*, for more information.

# Format Menu

This section contains a detailed description of the **Format** menu for the Variables and Watch windows. Figure 6-14 shows the **Format** menu.

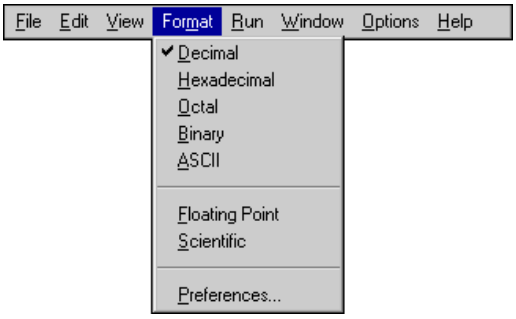


**Figure 6-14.** Format Menu

Use the commands in the **Format** menu to choose the format the Variables window uses to display numbers. You can change the format for an individual variable as well as the default formats for all variables. The first five items in the menu specify the available formats for displaying individual integers in the Variables window. You can display integers in decimal, hexadecimal, octal, binary, or ASCII format. The next two items in the **Format** menu specify the formats available for displaying individual real numbers. Real numbers appear in either floating-point or scientific notation. The last item, **Preferences**, sets the default formats for all integers and all real numbers.

# Run Menu

The **Run** menu contains the following subset of the commands that appear in the **Run** menu of the Source window as follows:

**Run Project**
**Continue**
**Step Over**
**Step Into**
**Finish Function**
**Terminate Execution**
**Break at First Statement**
**Breakpoints**

Refer to the *Run Menu* section in Chapter 4, *Source, Interactive Execution, and Standard Input/Output Windows*, for descriptions of each of these commands.

# Window Menu

The **Window** menu in the Variables and Watch windows operates the same way as in the Project window. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for command descriptions.

# Options Menu

This section contains a detailed description of the **Options** menu for the Variables and Watch windows.

Figure 6-15 shows the **Options** menu.

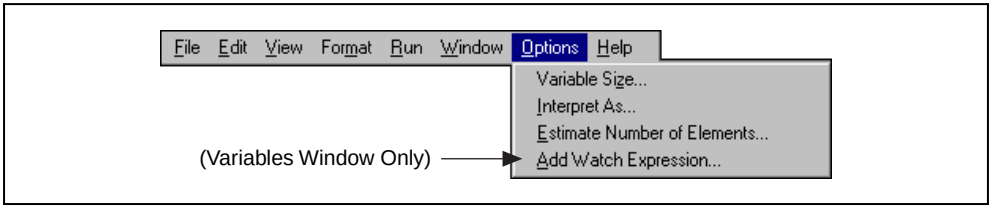


**Figure 6-15.** Options Menu

To use these commands, select a particular variable by clicking on it with the mouse or using the up and down arrow keys. Then access the command from the **Options** menu.

## 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).

## 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.

## 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. Refer to the *Limitations of User Protection* section in Chapter 1, *LabWindows/CVI Compiler,* of the *LabWindows/CVI Programmer Reference Manual* for more information about variable types that do not have user protection.

## Add Watch Expression. . .

**Add Watch Expression** invokes the Add/Edit Watch Expression dialog box from the Variables window. The *Watch Window* section earlier in this chapter explains this dialog box.

# 7

# Array and String Display Windows

This chapter describes the Array and String Display windows. Use these windows to inspect and modify the contents of a single array or string during a breakpoint.

☞ **Note** ***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, you can see beyond the NUL byte by using the*** **Display Entire Buffer** ***command in the*** **Options** ***menu.***

## Array Display Window

You can use the Array Display window to view and edit the contents of an array or string.

From the Variables window, use the **View»Array Display** command to invoke the Array Display window for the currently highlighted array. Also, you can double-click on an array to invoke the Array Display window.

Use the **View Variable Value** command in the **Run** menu of a Source window or the **Code** menu of a Function Panel window to invoke the Array Display window when the name of an array variable is under the keyboard cursor or is in the active function panel control.

Figure 7-1 shows the Array Display window for a single-dimensional array.

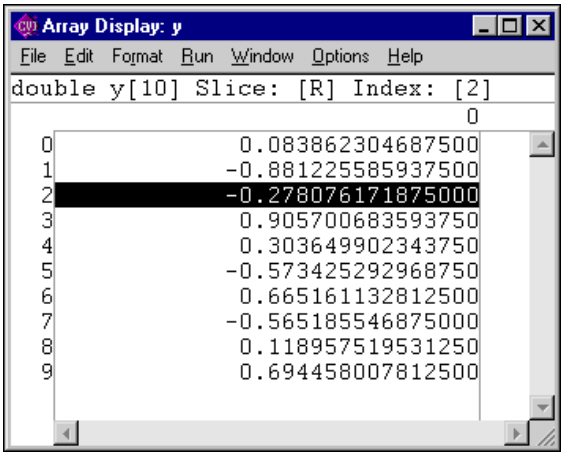


**Figure 7-1.** Array Display for a Double-Precision Array

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. Figure 7-2 shows the Array Display for a three-dimensional array.

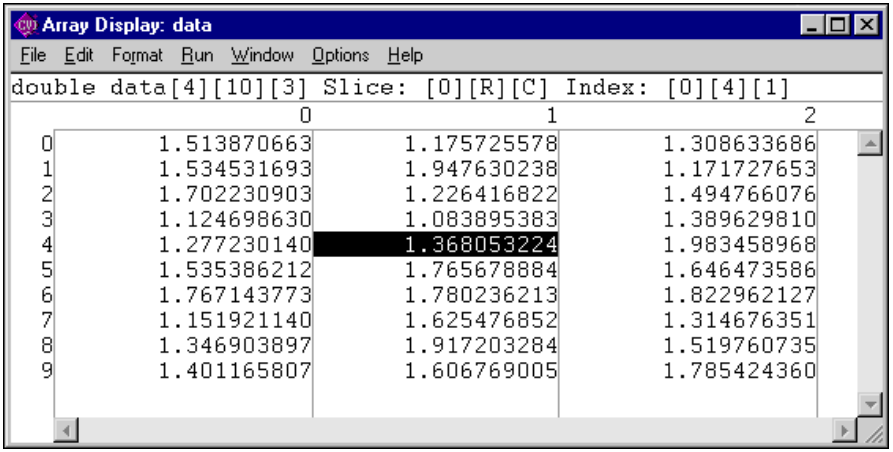


**Figure 7-2.** Array Display for a Three-Dimensional Array

The Array Display window shows a 2D 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. Use the **Reset Indices** command from the **Options** menu 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, as shown in Figure 7-3.

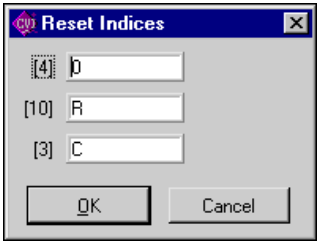


**Figure 7-3.** Reset Indices Dialog Box for a Three-Dimensional Array

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. For the three-dimensional array shown in Figure 7-2, there is one remaining dimension.

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, select **Cancel**.

# String Display Window

You can use the String Display window to view and edit the contents of a string variable or string array.

From the Variables window, use the **String Display** command in the **View** menu to invoke the String Display window for the currently highlighted string variable. Double-click on a string to invoke the String Display window.

From a Source or Function Panel window, use the **View Variable Value** command in the **Run** or **Code** menu to invoke the String Display window when the name of a string variable is under the keyboard cursor or is in the active function panel control.

Figure 7-4 shows the String Display for a string variable.

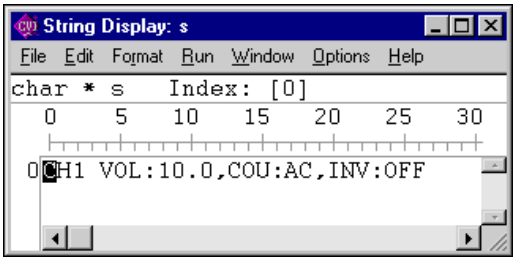


**Figure 7-4.** String Display for a String Variable

## Multi-Dimensional String Array

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

This section contains a detailed description of the **File** menu for the Array and String Display windows.

Figure 7-5 shows the **File** menu.

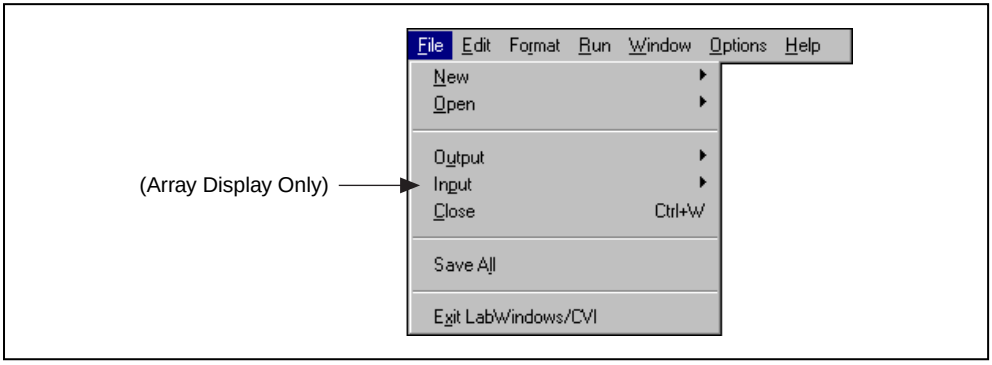


**Figure 7-5.** File Menu

## New

The **New** command operates the same way as in the Project window. For a description of this command, Refer to the discussion of the **New** command in the *File Menu* section of Chapter 3, *Project Window*.

## Open

The **Open** command operates the same way as in the Project window. For a description of this command, Refer to the discussion of the **Open** command in the *File Menu* section of Chapter 3, *Project Window*.

## 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 to prompt you to specify the name of the file.

## Input (Array Display Only)

Use the **Input** command to select an ASCII or binary data file on disk to replace the currently viewed array in memory.

## Close

The **Close** command closes the window.

## Save All

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

## Exit LabWindows/CVI

The **Exit LabWindows/CVI** command closes the current LabWindows/CVI session. If you have modified any open files since the last save, or if any windows contain unnamed files, LabWindows/CVI prompts you to save them to disk.

# Edit Menu for the Array Display Window

Figure 7-6 shows the **Edit** menu for the Array Display window.

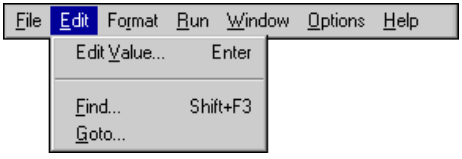


**Figure 7-6.** Edit Menu for the Array Display Window

## Edit Value. . .

The **Edit Value** command in the Array Display window invokes a dialog box that you can use to change the value of the selected array element.

## Find. . .

The **Find** command invokes the dialog box in Figure 7-7.

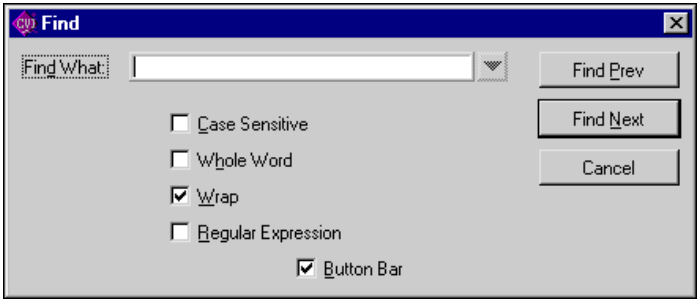


**Figure 7-7.** Find Dialog Box in the Array and String Display Windows

The **Find** command operates the same way as in the Variables window, but with fewer options. Refer to the *Find. . .* section in Chapter 6, *Variables and Watch Windows*, for information on how to use options in the Find dialog box.

### 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 desired location. For a single string, you specify only the column.

## Edit Menu for the String Display Window

Figure 7-8 shows the **Edit** menu for the String Display window.

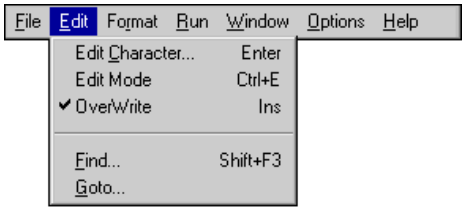


**Figure 7-8.** Edit Menu for the String Display Window

### Edit Character. . .

Use the **Edit Character** command in the String Display window to change one character at a time.

### 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.

### 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.

## Find. . .

The **Find** command operates the same way as in the Array Display window.

## Goto. . .

The **Goto** command operates the same way as in the Array Display window.

# Format Menu

This section contains a detailed description of the **Format** menu for the Array and String Display windows.

Figure 7-9 shows the **Format** menu.

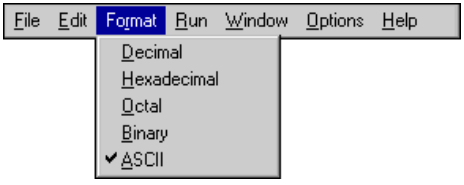


**Figure 7-9.** Format Menu

However, if a real array appears in the Array Display window, the **Format** menu appears as shown in Figure 7-10.

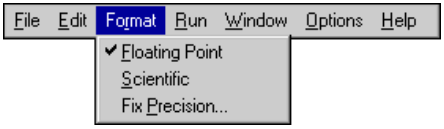


**Figure 7-10.** Format Menu for a Real Array in the Array Display Window

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

The **Run** menu contains a subset of the commands that appear in the **Run** menu of the Source window, as follows:

**Run Project**
**Continue**
**Step Over**
**Step Into**
**Finish Function**
**Terminate Execution**
**Break at First Statement**
**Breakpoints**

# Window Menu

The **Window** menu in the Array and String Display windows operates the same way as in the Project window. Refer to the *Window Menu* section in Chapter 3, *Project Window*, for command descriptions.

# Options Menu

This section contains a detailed description of the **Options** menu for the Array and String Display windows.

Figure 7-11 shows the **Options** menu.

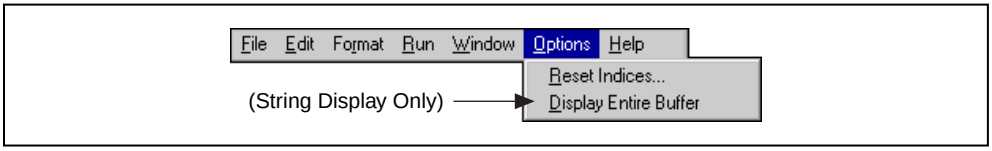


**Figure 7-11.** Options Menu

## Reset Indices. . .

Use **Reset Indices** in the Array Display window to set which array dimension appears as rows and which array dimension is displayed as columns.

Use **Reset Indices** in the String Display window to set which string array dimension appears as rows.

## Display Entire Buffer (String Display Only)

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**.

# A

# Source Window Keyboard Commands

The following table can help you quickly identify common Source window keyboard commands that are not in the menus.

**Table A-1.** Keyboard Help

| Category | Action | Shortcut Key(s) |
|---|---|---|
| Finding/Searching | Find again (up) | <Ctrl-F3> |
| | Find again (down) | <F3> |
| | Use selected text as search string | <Ctrl-Shift-F3> |
| | Replace selected text | <Ctrl-F11> |
| | Replace selected text and find again | <F11> |
| | Use selected text as replace string | <Ctrl-Shift-F11> |
| Windowing | Next window | <Ctrl-F6> |
| | Previous window | <Crtl-Shift-F6> |
| | Switch sub-windows | <F6> |

**Table A-1.** Keyboard Help (Continued)

| Category | Action | Shortcut Key(s) |
|---|---|---|
| Editing | Change text selection mode | <Ctrl-Ins> |
| | Toggle insert/overwrite mode | <Ins> |
| | Delete to end of line | <Ctrl-D> |
| | Backspace to beginning of word | <Ctrl-Shift-BkSp> |
| | Cut line to clipboard | <Ctrl-Y> |
| | Insert a new line above | <Shift-Enter> |
| | Insert a new line below | <Ctrl-Enter> |
| | Select text | <Shift-*arrow key*> |
| | Remove text selection | <Esc> |
| Cursor Movement | Up 1 line | <Up arrow> |
| | Down 1 line | <Down arrow> |
| | Left 1 column | <Left arrow> |
| | Right 1 column | <Right arrow> |
| | Scroll up 1 line | <Ctrl-Up> |
| | Scroll down 1 line | <Ctrl-Down> |
| | Left 1 word | <Ctrl-Left> |
| | Right 1 word | <Ctrl-Right> |
| | Top of window | <Ctrl-PgUp> |
| | Bottom of window | <Ctrl-PgDown> |
| | Beginning of line | <Home> |
| | End of line | <End> |
| | Move up 1 page | <PgUp> |
| | Move down 1 page | <PgDown> |
| | Top of file | <Ctrl-Home> |
| | Bottom of file | <Ctrl-End> |

# B

# Customer Communication

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

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

## Electronic Services

### Bulletin Board Support

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

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

### FTP Support

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

### Fax-on-Demand Support

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

### E-Mail Support (Currently USA Only)

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

`support@natinst.com`

## Telephone and Fax Support

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

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

# Technical Support Form

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

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

Name _____

Company _____

Address _____

_____

Fax ( ___ ) _____Phone ( ___ ) _____

Computer brand_____ Model _____Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed_____

Hard disk capacity _____MB Brand_____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem: _____

_____

_____

_____

_____

# LabWindows/CVI Hardware and Software Configuration Form

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

## National Instruments Products

Hardware revision _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

National Instruments software _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

## Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

# Documentation Comment Form

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

**Title:** *LabWindows/CVI User Manual*

**Edition Date:** February 1998

**Part Number:** 320681D-01

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

_____

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

E-Mail Address _____

Phone ( ___ ) _____ Fax ( ___ ) _____

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

**Fax to:** Technical Publications
National Instruments Corporation
512 794 5678

# Glossary

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

## A

active window — The window that user input affects at a given moment. The title of an active window is highlighted.

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

auto-exclusion — A mechanism that prevents preexisting 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 upon the position of the switch.

bps — bits per second

breakpoint — An interruption in the execution of a program.

Breakpoint command — A specific command that interrupts the execution of a program.

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

## C

| | |
|---|---|
| checkbox | A dialog box item that allows you to toggle between two possible options. |
| clipboard | A temporary storage area LabWindows/CVI uses to hold text that is cut, copied, or deleted 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. |

## 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 window | The window in which you build a function panel. It is described in the *LabWindows Instrument Driver Developers Guide*. |
| 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 window | The window in which you build the skeleton of a function panel file. It is described in the *LabWindows Instrument Driver Developers Guide.* |

## 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. |

## 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 (!). |
| in | inches |
| 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

| | |
|---|---|
| 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. |

## P

| | |
|---|---|
| Project window | A window containing a list of files your application uses. |
| 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

| | |
|---|---|
| s | seconds |
| 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 provide 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 upon the position of the cross-bar on the switch. |
| slider | The cross-bar on the slide control which 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, Analysis, Data Formatting and I/O, GPIB, GPIB-488.2, DDE, TCP RS-232, Utility, and C system libraries. |
| 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 or view text. |

## U

| | |
|---|---|
| User Interface Editor window | The window in which you build pull-down menus, dialog boxes, panels, and controls and save them to a User Interface Resource (`.uir`) file. It is described in the *LabWindows/CVI User Interface Reference Manual*. |

## 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. |

# Index

## Special Characters


_CVI_ macro, 3-60
_CVI_DEBUG macro, 3-60
_CVI_DLL_ macro, 3-60
_CVI_EXE_ macro, 3-60
_CVI_LIB_ macro, 3-60
__DEFALIGN macro, 3-60
__FLAT__ macro, 3-60
_M_IX86 macro, 3-60
_NI_BC macro, 3-60
_NI_i386_ macro, 3-60
_NI_mswin_ macro, 3-60
_NI_mswin16_ macro, 3-60
_NI_mswin32_ macro, 3-60
_NI_SC macro, 3-60
_NI_sparc_ macro, 3-60
_NI_unix_ macro, 3-60
_NI_VC macro, 3-60
_NI_WC macro, 3-60
__NT__ macro, 3-60
_WIN32 macro, 3-60
__WIN32__ macro, 3-60
_WINDOWS macro, 3-60


## A


About LabWindows/CVI command, Help menu, 4-42
activate option, UNIX, 1-8
Activate Panels When Resuming option, Run menu, 4-32 to 4-33
ActiveX Automation command, Library menu, 3-44
ActiveX Automation Library, 3-44. *See also* Tools menu.
Add File to Project command, File menu, 4-8
Add .FP File to Project command, File menu, 5-8
Add Missing Includes command, Build menu, 4-25
Add Program File to Project command, File menu, 5-8
Add Watch Expression command
    Code menu, 5-7, 5-19
    Edit menu, 6-9
    Options menu, 6-4, 6-15
    Run menu, 4-33
Add/Edit Watch Expression dialog box, 6-4 to 6-5
Advanced Analysis command, Library menu, 3-42
Alphabetize command, Select Function Panel dialog box, 3-40
Analysis command, Library menu, 3-42
Analysis Library, 3-42
ANSI C command, Library menu, 3-45
ANSI C Library, 3-45
appFont option, 1-8
applications, creating, 2-5 to 2-6
Array Display command, View menu, 6-13, 7-1
Array Display window
    displayed in View menu, 3-54
    Edit menu, 7-6 to 7-7
    File menu, 7-5 to 7-6
    Format menu, 7-8
    invoking, 7-1
    multi-dimensional arrays
        illustration, 7-3
        Reset Indices dialog box, 7-3
        specifying dimensions, 7-2 to 7-3
    Options menu, 7-9 to 7-10
    purpose and use, 2-4, 7-1
    Run menu, 7-9

single-dimensional array (figure), 7-2
Window menu, 7-9
Attach and Edit Source command, Edit Instrument dialog box, 3-39
attribute constants, selecting, 5-13 to 5-15
Auto Save Project command, File menu, 3-5 to 3-6

## B

Balance command, Edit menu, 4-12
Beginning/End of Selection command, View menu, 4-20
bin directory (table), 1-5
binary control parameters, specifying, 5-6
Bracket Styles command, Options menu, 4-38
brackets
finding pairs of, 4-12
setting location for, 4-38
Break at First Statement command, Run menu
Project window, 3-31
Source, Interactive Execution, and Standard Input/Output windows, 4-27, 4-30
Break on Library Errors option, 3-62
Breakpoint function, 4-26
breakpoints. *See also* watch variables/expressions.
applicable only in source code modules (note), 4-27
breakpoint state, 4-27
conditional, 4-28
Edit Breakpoint dialog box, 4-31 to 4-32
purpose and use, 4-26 to 4-27
resuming execution, 4-27
setting and clearing, 4-27
Breakpoints command, Run menu, 3-31, 4-27, 4-31 to 4-32
Breakpoints dialog box. *See also* Edit Breakpoint dialog box.
Add/Edit Item button, 4-31
Delete Item button, 4-32
Disable All button, 4-32
Enable All button, 4-32
Go to Line button, 4-32
illustration, 4-31
OK button, 4-32
Bring Standard Input/Output window to front option, 3-64
Browse Type Library dialog box, 3-47 to 3-52
Cancel button, 3-51
Context Sensitive Help buttons, 3-50
Function Prototype, 3-50
Generate button, 3-51
Help button, 3-51
illustration, 3-49
Instruments Prefix, 3-49
Method Description, 3-50
Method Names Too Long, 3-50
Method Tag, 3-50
Methods of *Object*, 3-50
Methods/Properties Pop-up Menu Ring, 3-50
Object Description, 3-50
Object Tag, 3-49
Objects in ActiveX Automation Server, 3-49
Properties of *Object*, 3-50
Properties Tag, 3-50
Property Description, 3-50
Property Names Too Long, 3-50
Select Target File dialog box, 3-51
Target File button, 3-51
Build Error window, 3-59
build errors
Build Errors in Next File command, 4-25
Next Build Error command, 4-25
Previous Build Error command, 4-25
Build Errors command, Window menu, 3-54
Build Errors in Next File command, View menu, 4-25

Build menu
Project window
Build Project command, 3-10
Compile File command, 3-10
Create Distribution Kit command, 3-24 to 3-30
Create Dynamic Link Library command, 3-15 to 3-18
Create Standalone Executable command, 3-13 to 3-15
Create Static Library command, 3-18 to 3-19
DLL Debugging, 3-19 to 3-21
DLL Debugging command, 3-19 to 3-21
External Compiler Support command, 3-21 to 3-24
illustration, 3-10
Instrument Driver Support Only command, 3-12 to 3-13
Link Project command, 3-11
Mark All for Compilation command, 3-11
Mark File for Compilation command, 3-11
Target command, 3-11 to 3-12
Update Program Files from Disk command, 3-11
Source, Interactive Execution, and Standard Input/Output windows
Add Missing Includes command, 4-25
Build Project command, 4-24
Clear Interactive Declarations command, 4-4, 4-24
Compile File command, 4-23 to 4-24
Generate Prototypes command, 4-25
illustration, 4-23
Insert Include Statements command, 4-24
Link Project command, 4-24
Mark File for Compilation command, 4-24
Next Build Error command, 4-25
Previous Build Error command, 4-25
Build Project command, Build menu
Project window, 3-10
Source, Interactive Execution, and Standard Input/Output windows, 4-24
bulletin board support, B-1
Button Bar option, Find command
Array and String Display windows, 7-6
Source, Interactive Execution, and Standard Input/Output windows, 4-16 to 4-17
Variables window, 6-8
buttons, adding and positioning on toolbar, 4-2

## C

calling convention, default, 3-57
Cascade Windows command, Window menu, 3-53
Case Sensitive option, Find command
Array and String Display windows, 7-6
Source, Interactive Execution, and Standard Input/Output windows, 4-14
Variables window, 6-7
CatchProtectionFaults option, 1-7
cfgdir configuration option, 1-4
Change Format command, Options menu, 5-23
Character Select mode, 4-6
Check Disk Dates Before Each Run option, 3-62 to 3-63
child structure, 6-3
child structure pointer in chain (figure), 6-12
Class Help dialog box, 3-41
Clear Interactive Declarations command
Build menu, 4-4, 4-24
Code menu, 5-11
Clear Tags command, View menu, 4-20

Clear Window command, Edit menu, 4-4, 4-11
Close All command, Window menu, 3-53
Close command, File menu
Array and String Display windows, 7-6
Function Panel windows, 5-8
Close Libraries command
Code menu, 5-11
Run menu, 4-30
Close Variable command, View menu, 6-3, 6-11
Closed Array, Variables window (figure), 6-10
code. *See* source files.
Code menu, Function Panel windows
Add Watch Expression command, 5-7, 5-19
Clear Interactive Declarations command, 5-11
Close Libraries command, 5-11
Declare Variable command, 5-4, 5-6, 5-10 to 5-11
illustration, 5-9
Insert Function Call command, 5-18
Run Function Panel command, 5-9
Select UIR Constant command, 5-11 to 5-12
Select Variable command, 5-15 to 5-18
Set Target File command, 5-18
View Variable Value command, 5-7, 5-19, 6-1, 7-4
code modules
adding to projects, 3-7
listing in Project window, 2-6
color coding tokens in source and include files, 4-39
Colors command, Options menu
Project window, 3-69
Source, Interactive Execution, and Standard Input/Output windows, 4-38
Column Select mode, 4-7
Command Line command, Options menu, 3-63
common control function panel, 5-7
comparing source files. *See* Diff command, Edit menu.
Compatibility with option, 3-57
compile errors, maximum number of, 3-57
Compile File command, Build menu
Project window, 3-10
Source, Interactive Execution, and Standard Input/Output windows, 4-23 to 4-24
compiled files, 2-6
compiler defines
predefined macros, 3-60 to 3-61
syntax, 3-59
compiler options
Compatibility with, 3-57
Default Calling Convention, 3-57
Display status dialog during build, 3-59
Enable signed/unsigned pointer mismatch warning, 3-58
Enable unreachable code warning, 3-59
Maximum number of compile errors, 3-57
Prompt for include file paths, 3-59
Require function prototypes, 3-57 to 3-58, 4-4
Require return values for non-void functions, 3-58
Show Build Error window for warnings, 3-59
Stop on first file with errors, 3-59
Compiler Options command, Options menu, 3-57 to 3-59
compiler support, external. *See* External Compiler Support dialog box.
compiling files. *See* Build menu.
conditional breakpoints, 4-28
configuration options
cfgdir, 1-4
cvidir, 1-4 to 1-5
resdir, 1-5
tmpdir, 1-5

UNIX, 1-3
Windows 3.1, 1-3
Windows 95/NT, 1-2
constants, user interface. *See* user interface constants, selecting.
Contents command, Help menu, 4-42
context menus, Source window, 4-3
Continue command, Run menu
Project window, 3-31
Source, Interactive Execution, and Standard Input/Output windows, 4-29
Control command, Help menu, 5-24
<Control> key (SPARCstation), 3-64
Copy command, Edit menu, 4-11
Create ActiveX Automation Controller command, Tools menu
Project window, 3-46 to 3-52
Browse Type Library dialog box, 3-47 to 3-52
Select ActiveX Automation Server dialog box, 3-47
updating existing controller instrument drivers, 3-52
Source, Interactive Execution, and Standard Input/Output windows, 4-34
Create Distribution Kit command, Build menu, 3-24
Create Distribution Kit dialog box, 3-25 to 3-30
Advanced Distribution Kit options, 3-29 to 3-30
Executable to Run After Setup option, 3-30
Installation Titles option, 3-30
Build Information section, 3-25 to 3-27
File Groups section, 3-27 to 3-28
illustration, 3-25
Installation Script File section, 3-29
Main section, 3-28
Create Dynamic Link Library command, Build menu, 3-15
Create Dynamic Link Library dialog box, 3-15 to 3-18
Cancel button, 3-17
DLL file field, 3-16
Exports
Change button, 3-17
Export What field, 3-17
Include File Symbols option, 3-17
Symbols Marked for Export option, 3-17
Import Library Base Name field, 3-16
Import Library Choices button, 3-16 to 3-17
OK button, 3-17
Prompt before overwriting file checkbox, 3-16
Type Library button, 3-17
Using LoadExternalModule
Add Files to DLL button, 3-17
Help button, 3-17
Where to Copy DLL control, 3-16
Create IVI Instrument Driver command, Tools menu
Project window, 3-52
Source, Interactive Execution, and Standard Input/Output windows, 4-35
Create Object File command, Options menu, 4-41 to 4-42
Create Standalone Executable command, Build menu, 3-13 to 3-15
Create Standalone Executable dialog box, 3-14 to 3-15
Application Executable File field, 3-14
Application Icon File field, 3-14
Application Title field, 3-14
Cancel button, 3-15
Icon control, 3-14
OK button, 3-15
Prompt before overriding executable file checkbox, 3-14

Using LoadExternalModule item, 3-14 to 3-15
Version Info button, 3-15
Create Static Library command, Build menu, 3-18 to 3-19
Create Static Library dialog box, 3-18 to 3-19
Cancel button, 3-18
Library File field, 3-18
Library Generation Choices button, 3-18
OK button, 3-18
Prompt before overwriting file checkbox, 3-18
creating
applications, 2-5 to 2-6
object files, 4-41 to 4-42
standalone executables. *See* standalone executables, creating and distributing.
user interface, 2-6
Windows DLLs. *See* Windows DLLs.
curly braces
finding pairs of, 4-12
setting location for, 4-38
Current Tree command, View menu, 5-20
customer communication, *xxiv*, B-1 to B-2
customizing
bracket styles, 4-38
colors, 3-69
fonts, 1-6 to 1-7, 3-69, 4-38
toolbars, 4-2 to 4-3
Cut command, Edit menu, 4-11
_CVI_ macro, 3-60
_CVI_DEBUG macro, 3-60
_CVI_DLL_ macro, 3-60
_CVI_EXE_ macro, 3-60
_CVI_LIB_ macro, 3-60
cvidir configuration option, 1-4 to 1-5


## D


Data Acquisition command, Library menu, 3-43
Data Acquisition Library
definition, 3-43
purpose and use, 2-3
data type compatibility for function panel variables, 5-17 to 5-18
date option, DSTRules, 1-6
dates
Check Disk Dates Before Each Run option, 3-62 to 3-63
displaying files in chronological order, 3-9
displaying for project list files, 3-9
daylight savings time, setting, 1-6
DDE command, Library menu, 3-44
DDE Library, 3-44
debug options
CatchProtectionFaults, 1-7
DisplayCVIDebugVxDMissingMessage, 1-7
LoadCVIDebugVxD, 1-7
debugging DLLs. *See* DLL Debugging command.
debugging levels
Extended, 3-62
No Run-time Checking, 3-62
None, 3-62
Standard, 3-62
Declare Variable command, Code menu, 5-10 to 5-11
specifying input control parameter, 5-4
specifying output control parameter, 5-6
Declare Variable dialog box, 5-10 to 5-11
Add declaration to current block in target file checkbox, 5-10
Add declaration to top of target file checkbox, 5-10
Cancel button, 5-10
Execute declaration checkbox, 5-10
illustration, 5-10
Number of Elements box, 5-10
OK button, 5-10

Variable Name text box, 5-10
Variable Type message, 5-10
__DEFALIGN macro, 3-60
Default All command, Options menu, 5-22
Default calling convention option, 3-57
Default Control command, Options menu, 5-22
Delete command, Edit menu, 4-11
Delete Watch Expression command, Edit menu, 6-9
Detach Program command, Edit Instrument dialog box, 3-39
dialogFont option, 1-8
DialogFontBold option, 1-7
DialogFontName option, 1-6
DialogFontSize option, 1-7
Diff command, Edit menu, 4-13
Diff With, 4-13
Find Next Difference, 4-13
Ignore White Space, 4-13
Match Criteria, 4-13
Recompare Selections Ignoring White Space, 4-13
Synchronize at Top, 4-13
Synchronize Selections, 4-13
directory configuration option, 1-4 to 1-5
Display Entire Buffer command, Options menu, 7-10
display options, UNIX
activate, 1-8
useDefaultColors, 1-9
Display status dialog during build option, 3-59
DisplayCVIDebugVxDMissingMessage option, 1-7
distributing standalone executables. *See* standalone executables, creating and distributing.
Distribution Kit. *See* Create Distribution Kit dialog box.
DLL Debugging command, 3-19 to 3-21
location of required files, 3-19 to 3-20
multithreaded executables, 3-21
running external process, 3-21
running program in LabWindows/CVI, 3-20 to 3-21
DLLs. *See* Windows DLLs.
documentation
conventions used in manual, *xxii-xxiii*
LabWindows/CVI documentation set, *xxiii-xxiv*
organization of manual, *xxi-xxii*
Down Call Stack command, Run menu, 4-33
DSTRules option, 1-6
Dynamic Memory command, Run menu, 4-33
dynamic-link library files, required in project file list, 3-1


## E


Easy I/O for DAQ command, Library menu, 3-43
Easy I/O for DAQ Library
definition, 3-43
purpose and use, 2-3
Edit Breakpoint dialog box, 4-31 to 4-32
Edit Character command, Edit menu, 7-7
Edit command, Instrument menu, 3-38 to 3-39. *See also* Edit Instrument dialog box.
Edit Function Panel Window command
Options menu, 5-23
Tools menu, 4-36
Edit Function Tree command
Edit Instrument dialog box, 3-39
Tools menu, 4-35
Edit Instrument Attributes command, Tool menu, 4-35
Edit Instrument dialog box
Attach and Edit Source command, 3-39
Detach Program command, 3-39
Edit Function Tree command, 3-39
illustration, 3-38

Reattach Program command, 3-39
Show Info command, 3-39
Edit menu
Array Display window
Edit Value command, 7-6
Find command, 7-6 to 7-7
Goto command, 7-7
illustration, 7-6
Project window, 3-6 to 3-8
Add Files to Project command, 3-7
Exclude File from Build command, 3-8
illustration, 3-6
Include File in Build command, 3-8
Move Item Down command, 3-8
Move Item Up command, 3-8
Remove File command, 3-8
Source, Interactive Execution, and Standard Input/Output windows
Balance command, 4-12
Clear Window command, 4-4, 4-11
Copy command, 4-11
Cut command, 4-11
Delete command, 4-11
Diff command, 4-13
disabled commands (note), 4-10
Find command, 4-14 to 4-17
Go to Definition command, 4-13
illustration, 4-10
Insert Construct command, 4-12
Next File command, 4-18
Paste command, 4-11
Redo command, 4-10
Replace command, 4-17 to 4-18
Resolve All Excluded Lines command, 4-12
Select All command, 4-11
Toggle Exclusion command, 4-4, 4-12
Undo command, 4-10
String Display window
Edit Character command, 7-7
Edit Mode command, 7-7
Find command, 7-8
Goto command, 7-8
Overwrite command, 7-7
Variables window
Edit Value command, 6-7
Find command, 6-6 to 6-8
illustration, 6-6
Next Scope command, 6-8
Previous Scope command, 6-8
Watch window
Add Watch Expression command, 6-9
Delete Watch Expression command, 6-9
Edit Value command, 6-9
Edit Watch Expression command, 6-9
Find command, 6-9
illustration, 6-9
Edit Mode command, Edit menu, 7-7
Edit Value command, Edit menu
Array Display window, 7-6
Variables window, 6-7
Watch window, 6-9
Edit Watch Expression command, Edit menu, 6-9
Editor Preferences command, Options menu
dialog box, 4-37 to 4-38
Line Terminator option, 4-38
Paste option, 4-37
Tabs option, 4-38
Undo option, 4-10, 4-37
editorFont option, 1-8
electronic support services, B-1 to B-2
e-mail support, B-2
Enable signed/unsigned pointer mismatch warning option, 3-58
Enable unreachable code warning option, 3-59

End of Selection command, View menu, 4-20
Environment command, Options menu, 3-40, 3-63
environment options
  Bring Standard Input/Output window to front whenever modified, 3-64
  Maximum number of lines in Standard Input/Output window, 3-64
  Sleep policy when not running program, 3-63
  Use host's system standard I/O, 3-64
  Use only one function panel window, 3-40, 3-64
  Warp mouse over dialog boxes, 3-64
Error command, View menu, 5-19
errors
  Break on library errors option, 3-62
  build errors, 4-25
  Display status dialog during build option, 3-59
  Maximum number of compile errors option, 3-57
  run-time error reporting, 3-31, 4-29
  Show Build Error window for warnings option, 3-59
  Stop on first file with errors option, 3-59
  terminating compilation for file errors, 3-59
Estimate Number of Elements command, Options menu, 6-15
Exclude File from Build command, Edit menu, 3-8
Exclude Function command, Options menu, 5-22
excluding lines of code, 4-4, 4-12
executables, creating and distributing. *See* standalone executables, creating and distributing.
Execute command, Run menu, 3-32
Exit LabWindows/CVI command, File menu
  Array and String Display windows, 7-6
  Function Panel windows, 5-8
  Project window, 3-6
  Source, Interactive Execution, and Standard Input/Output windows, 4-9
  Variables and Watch windows, 6-6
Expand Variable command, View menu, 6-3, 6-10 to 6-11
Expanded Array, Variables window (figure), 6-11
expressions. *See* watch variables/expressions.
expressions, regular (table), 4-15 to 4-16
External Compiler Support command, 3-21
External Compiler Support dialog box, 3-22 to 3-24
  ANSI C Library field, 3-23
  CVI Libraries field, 3-23
  illustration, 3-22
  Other Symbols checkmark, 3-23
    Header File field, 3-23
    Object File field, 3-23
  UIR Callbacks Object File option, 3-22
  Using LoadExternalModule to Load Object and Static Library Files optimization, 3-22 to 3-23
external process
  debugging DLLs, 3-21
  selecting, 3-32

## F

fax and telephone support numbers, B-2
Fax-on-Demand support, B-2
file extensions, displaying project files in order of, 3-9
File menu
  Array and String Display windows
    Close command, 7-6
    Exit LabWindows/CVI command, 7-6
    illustration, 7-5
    Input command, 7-5

New command, 7-5
Open command, 7-5
Output command, 7-5
Save All command, 7-6
Function Panel windows
Add .FP File to Project command, 5-8
Add Program File to Project command, 5-8
Close command, 5-8
Exit LabWindows/CVI command, 5-8
illustration, 5-7
New command, 5-7
Open command, 5-8
Save All command, 5-8
Project window, 3-3 to 3-6
Auto Save Project command, 3-5 to 3-6
Exit LabWindows/CVI command, 3-6
illustration, 3-3
most recently closed files list, 3-6
New command, 3-4
Open command, 3-4 to 3-5
Print command, 3-6
Save command, 3-5
Save All command, 3-5
Save As command, 3-5
Source, Interactive Execution, and Standard Input/Output windows
Add File to Project command, 4-8
Close command, 4-8
Exit LabWindows/CVI command, 4-9
Hide command (note), 4-8
illustration, 4-7
New command, 4-7
Open command, 4-8
Open Quoted Text command, 4-8
Print command, 4-9
Read Only command, 4-9
Save command, 4-8
Save All command, 4-8
Save As command, 4-8
Save Copy As command, 4-8
Variables and Watch windows
Exit LabWindows/CVI command, 6-6
Hide command, 6-6
illustration, 6-5
New command, 6-6
Open command, 6-6
Output command, 6-6
Save All command, 6-6
<filename> startup option (table), 1-1
files. *See also* project files.
adding to project list, 3-7
Check Disk Dates Before Each Run option, 3-62 to 3-63
format conversion when loading, 3-38
instrument driver files, 3-33 to 3-34
Find command, Edit menu
Array Display window, 7-6 to 7-7
Source, Interactive Execution, and Standard Input/Output windows, 4-14 to 4-17
String Display window, 7-8
Variables window, 6-6 to 6-8
Watch window, 6-9
Find dialog box
Array Display window, 7-6 to 7-7
Source, Interactive Execution, and Standard Input/Output windows, 4-14 to 4-17
Variables window, 6-6 to 6-8
Find Function Panel command, View menu
Function Panel windows, 5-20
Source, Interactive Execution, and Standard Input/Output windows, 4-22
Find Next option, Find command
Array Display window, 7-6

Source, Interactive Execution, and Standard Input/Output windows, 4-17
Variable Display and Watch Windows, 6-8
Find Prev option, Find command
Array Display window, 7-6
Source, Interactive Execution, and Standard Input/Output windows, 4-17
Variable Display and Watch Windows, 6-8
Find UI Objects command, View menu, 4-23
Find What text box option, Find command, 4-14
Finish Function command, Run menu, 4-30
First Function Panel Window command, View menu, 5-21
First Panel command, View menu, 5-2
Fixup pathnames when project is moved option, 3-68
__FLAT__ macro, 3-60
Flatten option, Select Function Panel dialog box, 3-40, 5-2
Follow Pointer Chain command, View menu, 6-3, 6-11 to 6-12
Font command, Options menu
Project window, 3-69
Source, Interactive Execution, and Standard Input/Output windows, 4-38
font directory (table), 1-5
font options
appFont, 1-8
dialogFont, 1-8
DialogFontBold, 1-7
DialogFontName, 1-6
DialogFontSize, 1-7
editorFont, 1-8
menuFont, 1-8
messageBoxFont, 1-8
format conversion of files during loading, 3-38
Format menu
Array and String Display windows, 7-8
Variables and Watch windows, 6-13
Formatting and I/O command, Library menu, 3-45
Formatting and I/O Library, 3-45
.fp files. *See* instrument driver function panel (.fp) files.
FTP support, B-1
function classes, 5-1
Function command, Help menu, 5-24
Function Help dialog box, 3-41
function panel, recalling. *See* Recall Function Panel command, View menu.
function panel controls
binary control parameter, 5-6
common control panel, 5-7
global control, 5-7
illustration, 5-4
input control parameter, 5-5
numeric control parameter, 5-5
output control parameter, 5-6
overriding with Toggle Control Style command, 5-22 to 5-23
restoring default value, 5-22
return value control parameter, 5-4 to 5-5
ring control parameter, 5-6
slide control parameter, 5-5
types of controls (figure), 5-4
viewing arrays, structures, and variables, 5-7
Function Panel Editor window, 2-5
Function Panel Help Editor window, 2-5
Function Panel History command, View menu
Function Panel windows, 5-20
Source, Interactive Execution, and Standard Input/Output windows, 4-21
Function Panel Tree command, View menu, 4-21
Function Panel windows
Code menu, 5-9 to 5-19

displayed in Window menu, 3-55
File menu, 5-7 to 5-8
Generated Code Box, 5-3
Help menu, 5-23 to 5-24
illustration, 5-2
Instrument menu, 5-21
Library menu, 5-21
multiple function panels per window, 5-3
Options menu, 5-22 to 5-23
purpose and use, 2-4
View menu, 5-2, 5-19 to 5-21
Window menu, 5-21
function panels. *See also* function panel controls.
accessing, 5-1 to 5-2
from Instrument menu, 3-40 to 3-41
definition, 2-4, 5-1
executing in Interactive Execution window, 5-1
finding functions, 4-22
multiple function panels per window, 5-3
selecting, 5-1 to 5-2
function prototypes, enabling, 3-57 to 3-58
Function subwindow, Variables window, 6-2
Function Tree command, 3-7
Function Tree Editor window
opening
with New command, 3-4
with Open command, 3-5
purpose and use, 2-5
Function Tree Help Editor window, 2-5
function trees
definition, 5-1
files displayed in Window menu, 3-55

## G

Generate DLL Glue Object command, Options menu, 4-41
Generate DLL Glue Source command, Options menu, 4-40 to 4-41
Generate DLL Import Library command, Options menu, 4-40
Generate DLL Import Source command, Options menu, 4-39 to 4-40
Generate Prototypes command, Build menu, 4-25
Generate Visual Basic Include command, Options menu, 4-41
Generated Code Box, 5-3
global control, 5-7
Global subwindow, Variables window, 6-2
glue code. *See* Windows DLLs.
glue object, generating, 4-41
Go to Cursor command, Run menu, 4-29
Go to Definition command
Edit menu, 4-13
View menu, 6-12
Go to Execution Position command, View menu, 6-12
Goto command, Edit menu
Array Display window, 7-7
String Display window, 7-8
GPIB Library, 3-43
GPIB/GPIB 488.2 command, Library menu, 3-43

## H

header files
including, 3-7
optional in project file list, 3-1
Help dialog box, for functions, windows, or classes, 3-41
Help Editor windows, displayed in Window menu, 3-55
Help menu
Function Panel windows
Control command, 5-24
Function command, 5-24
illustration, 5-23

Source, Interactive Execution, and Standard Input/Output windows, 4-42
About LabWindows/CVI command, 4-42
Contents command, 4-42
Keyboard Help command, 4-42
Search for Help On command, 4-42
Visit LabWindows/CVI Web Page command, 4-42
Windows SDK command, 4-42
Hide command, File menu
Interactive Execution and Standard Input/Output windows (note), 4-8
Variables and Watch windows, 6-6
Hide Windows option, 3-62
host Standard I/O, selecting, 3-64
hyperhelp directory (table), 1-5

## I

icons
associated with variables, 6-2
Project window, 3-2 to 3-3
IEW. *See* Interactive Execution window.
include directory (table), 1-5
Include File command, View menu, 5-20
Include File in Build command, Edit menu, 3-8
include files
adding missing files, 4-25
generating for Visual Basic, 4-41
prompting for path, 3-59
tracking dependencies, 3-59
Include option, Add Files to Project command, 3-7
Include Paths command, Options menu, 3-61
Input command, File menu, 7-5
input control parameters, specifying, 5-5
Insert Construct command, Edit menu, 4-12
Insert Function Call command, Code menu, 5-18
Insert Include Statements command, Build menu, 4-24
instrsup.dll
functions from selected libraries, 3-12
linking project to selected libraries, 3-12
multithreaded safe functions, 3-13
Standard Input/Output Window not supported, 3-13
Utility Library functions in, 3-13
Instrument Directories command, Options menu, 3-61 to 3-62
instrument driver function panel (.fp) files
adding to project list, 5-8
dummy .fp files for support libraries, 3-45 to 3-46, 3-66
purpose and use, 3-1
Instrument Driver Support Only command, Build menu, 3-12 to 3-13
instrument drivers. *See also* IVI instrument drivers.
definition, 3-32
files for instrument drivers, 3-33 to 3-34
loading/unloading, 3-34 to 3-36
instruments without instrument program, 3-36
precedence rules, 3-34 to 3-35
modifying, 3-36 to 3-37
modules containing non-instrument functions, 3-36
programming, 3-1
VXI*plug&play* instrument driver files, 3-33 to 3-34
Instrument Library, 2-3 to 2-4
Instrument menu
accessing function panels, 5-1
Function Panel windows, 5-21
illustration, 3-32
Project window
accessing function panels, 3-40 to 3-41
Edit command, 3-38 to 3-39

illustration, 3-37
Load command, 3-37 to 3-38
loading user libraries, 3-45
Unload command, 3-38
Source, Interactive Execution, and Standard Input/Output windows, 4-34
Intelligent Virtual Instrument drivers. *See* IVI instrument drivers.
Intelligent Virtual Instrument Library. *See* IVI Library.
Interactive Execution command, Window menu, 3-55
Interactive Execution window, 4-3 to 4-5
Build menu, 4-23 to 4-25
Edit menu, 4-9 to 4-18
excluding lines, 4-12
executing code, 4-4 to 4-5
rules for, 4-4 to 4-5
executing function panels, 5-1
File menu, 4-7 to 4-9
Instrument menu, 4-34
Library menu, 4-34
Options menu, 4-36 to 4-42
purpose and use, 2-4
rules for executing code, 4-4 to 4-5
Run menu, 4-26 to 4-33
selecting text, 4-5 to 4-7
subwindows, 4-5
Tools menu, 4-34 to 4-36
View menu, 4-19 to 4-23
Window menu, 4-36
Interpret As command, Options menu, 6-15
IVI command, Library menu, 3-44
IVI instrument drivers
creating, 3-52, 4-35
editing attributes, 4-35
IVI Library
definition, 3-44
purpose and use, 2-3

## K

keyboard commands
bypassing Find dialog box, 4-17
bypassing Replace dialog box, 4-18
Source window (figure), A-1 to A-2
Keyboard Help command, Options menu, 4-42
keyboard option, UNIX (useMetaKey), 1-9
Keyboard Options command, Options menu, 3-64

## L

LabWindows/CVI. *See also* specific windows.
components, 2-1 to 2-5
Data Acquisition Library and Easy I/O for DAQ Library, 2-3
Instrument Library, 2-3
LabWindows/CVI environment, 2-4 to 2-5
standard libraries, 2-2
User Interface Library, 2-3
VISA Library and IVI Library, 2-3
creating applications, 2-5 to 2-6
environment, 2-4 to 2-5
Last Function Panel Window command, View menu, 5-21
Last Panel command, View menu, 5-2
libraries. *See also* standard libraries; user libraries; specific library names.
files required in project file list, 3-1
selected libraries in instrsup.dll, 3-12
Library menu
accessing function panels, 5-1
Function Panel windows, 5-21
Project window
ActiveX Automation command, 3-44
Analysis/Advanced Analysis command, 3-42
ANSI C command, 3-45
Data Acquisition command, 3-43
DDE command, 3-44

Easy I/O for DAQ command, 3-43
Formatting and I/O command, 3-45
GPIB/GPIB 488.2 command, 3-43
illustration, 3-42
installing user libraries, 3-45 to 3-46
IVI command, 3-44
RS-232 command, 3-43
TCP command, 3-44
User Interface command, 3-42
Utility command, 3-45
VISA command, 3-44
VXI command, 3-43
X Property command, 3-44
Source, Interactive Execution, and Standard Input/Output windows, 4-34
Library option, Add Files to Project command, 3-7
Library Options command, Options menu, 3-45, 3-64 to 3-66
Library Options dialog box
illustration, 3-65
National Instrument Libraries, 3-66
User Libraries list, 3-65 to 3-66
Line command, View menu, 4-20
Line Icons command, View menu, 4-19, 4-27
Line Numbers command, View menu, 4-19
Line Select mode, 4-6
Line Terminator option, Editor Preferences command, 4-38
lines of code
excluding, 4-4, 4-12
Maximum number of lines in Standard Input/Output window option, 3-64
Link Project command, Build menu
Project window, 3-11
Source, Interactive Execution, and Standard Input/Output windows, 4-24
Load command, Instrument menu, 3-37 to 3-38
LoadCVIDebugVxD option, 1-7
loading/unloading instrument drivers, 3-34 to 3-36
instruments without instrument program, 3-36
precedence rules, 3-34 to 3-35


## M


macros, for writing platform-dependent code, 3-60 to 3-61
manual. *See* documentation.
Mark All for Compilation command, Build menu, 3-11
Mark File for Compilation command, Build menu
Project window, 3-11
Source, Interactive Execution, and Standard Input/Output windows, 4-24
Maximum number of compile errors option, 3-57
Maximum number of lines in Standard Input/Output window option, 3-64
maximum stack size, setting, 3-62
menuFont option, 1-8
messageBoxFont option, 1-8
<Meta> key
SPARCstation, 3-64
useMetaKey option, UNIX, 1-9
Microsoft Visual Basic, generating include file for, 4-41
Microsoft Windows. *See* Windows.
Microsoft Windows DLLs. *See* Windows DLLS.
Minimize All command, Window menu, 3-53
_M_IX86 macro, 3-60
mouse
Warp Mouse Over Dialog Boxes environment option, 3-64
warpMouseOverDialogBoxes option, UNIX, 1-9
Move Item Down command, Edit menu, 3-8
Move Item Up command, Edit menu, 3-8

multi-dimensional arrays
illustration, 7-3
Reset Indices dialog box, 7-3
specifying dimensions, 7-2 to 7-3
multi-dimensional strings, 7-4
Multiple Files option, Find command, 4-16

## N

Name option, Find command, 6-8
National Instruments libraries. *See* standard libraries; specific libraries.
New command, File menu
Array and String Display windows, 7-5
Function Panel windows, 5-7
Project window, 3-4
Source, Interactive Execution, and Standard Input/Output windows, 4-7
Variables and Watch windows, 6-6
New Window option, Select Function Panel dialog box, 3-40
-newproject option (table), 1-2
Next Build Error command, View menu, 4-25
Next File command, Edit menu, 4-18
Next Function Panel command, View menu, 5-20
Next Function Panel Window command, View menu, 5-21
Next Panel command, View menu, 5-2
Next Scope command, Edit menu, 6-8
Next Tag command, View menu, 4-20
_NI_BC macro, 3-60
_NI_i386_ macro, 3-60
_NI_mswin_ macro, 3-60
_NI_mswin16_ macro, 3-60
_NI_mswin32_ macro, 3-60
_NI_SC macro, 3-60
_NI_sparc_ macro, 3-60
_NI_unix_ macro, 3-60
_NI_VC macro, 3-60
_NI_WC macro, 3-60
No Sorting command, View menu, 3-9
non-void functions, requiring return values for, 3-58
__NT__ macro, 3-60
NUL byte, difference from space character (note), 6-1, 7-1
numeric control parameters, specifying, 5-5

## O

object files
creating, 4-41 to 4-42
required in project file list, 3-1
Object option, Add Files to Project command, 3-7
one-dimensional array, displaying in Array Display window (figure), 7-2
Open command, File menu
Array and String Display windows, 7-5
Function Panel windows, 5-8
Project window, 3-4 to 3-5
Source, Interactive Execution, and Standard Input/Output windows, 4-8
Variables and Watch windows, 6-6
Open Quoted Text command, File menu, 4-8
Options menu
Array and String Display windows
Display Entire Buffer command, 7-10
illustration, 7-9
Reset Indices command, 7-2, 7-3, 7-4, 7-9
Function Panel windows
Change Format command, 5-23
Default All command, 5-22
Default Control command, 5-22
Edit Function Panel Window command, 5-23
Exclude Function command, 5-22
illustration, 5-22

Toggle Control Style command, 5-22
Toolbar command, 5-22
Project window
Colors command, 3-69
Command Line command, 3-63
Compiler Options command, 3-57 to 3-59
Environment command, 3-40, 3-63
Font command, 3-69
illustration, 3-56
Include Paths command, 3-61
Instrument Directories command, 3-61 to 3-62
Keyboard Options command, 3-64
Library Options command, 3-45, 3-65 to 3-66
Project Move Options command, 3-68 to 3-69
Run Options command, 3-62 to 3-63
Source, Interactive Execution, and Standard Input/Output windows
Bracket Styles command, 4-38
Colors command, 4-38
Create Object File command, 4-41 to 4-42
Editor Preferences command, 4-10, 4-37 to 4-38
Font command, 4-38
Generate DLL Glue Object command, 4-41
Generate DLL Glue Source command, 4-40 to 4-41
Generate DLL Import Library command, 4-40
Generate DLL Import Source command, 4-39 to 4-40
Generate Visual Basic Include command, 4-41
illustration, 4-36
Keyboard Help command, 4-42
Syntax Coloring option, 4-39
Toolbar command, 4-38
Translate DOS LW program command, 4-39
User Defined Tokens for Coloring command, 4-39
Variables and Watch windows
Add Watch Expression command, 6-4, 6-15
Estimate Number of Elements command, 6-15
illustration, 6-14
Interpret As command, 6-15
Variable Size command, 6-15
Output command, File menu
Array and String Display windows, 7-5
Variables and Watch windows, 6-6
output control parameters, specifying, 5-6
Overwrite command, Edit menu, 7-7


## P


parent structure, 6-3
parent structure pointer in chain (figure), 6-11
parentheses, finding pairs of, 4-12
Paste command, Edit menu, 4-11
Paste option, Editor Preferences command, 4-37
path options
Fixup pathnames when project is moved, 3-68
Prompt for include file paths, 3-59
pathnames
displaying project files by full pathname, 3-9
sorting project files by pathname, 3-9
paths for compiler, listing, 3-61
pointer mismatch warning, enabling, 3-58
-p*ProcessID* option (table), 1-2
predefined macros, for writing platform-dependent code, 3-60 to 3-61

Previous Build Error command, View menu, 4-25
Previous Function Panel command, View menu, 5-20
Previous Function Panel Window command, View menu, 5-21
Previous Panel command, View menu, 5-2
Previous Scope command, Edit menu, 6-8
Previous Tag command, View menu, 4-20
Print command, File menu
  Project window, 3-6
  Source, Interactive Execution, and Standard Input/Output windows, 4-9
Project command, Window menu, 3-54
project files
  optional files, 3-1
  options for displaying, 3-8 to 3-9
  required files, 3-1
  saving automatically, 3-5 to 3-6
Project Move Options command, Options menu
  description, 3-68 to 3-69
  Fixup pathnames when project is moved, 3-68
Project window
  Build menu, 3-10 to 3-30
  Edit menu, 3-6 to 3-8
  File menu, 3-3 to 3-6
  icons, 3-2 to 3-3
  illustration, 2-5, 3-2
  Instrument menu, 3-37 to 3-41
  Library menu, 3-41 to 3-46
  opening
    with New command, 3-4
    with Open command, 3-4 to 3-5
  optional files, 3-1
  Options menu, 3-56 to 3-69
  overview, 3-1 to 3-3
  purpose and use, 2-4
  required files, 3-1
  Run menu, 3-30 to 3-32
  Tools menu, 3-46 to 3-52
  View menu, 3-8 to 3-9
  Window menu, 3-53 to 3-56
Prompt for include file paths option, 3-59

## R

Read Only command, File menu, 4-9
Reattach Program command, Edit Instrument dialog box, 3-39
Recall Function Panel command, View menu
  invoking, 4-21
  multiple functions in one function panel window, 4-22
  multiple panels for one function, 4-22
  purpose, 4-21
  recalling from function name only, 4-21
  syntax requirements, 4-22
Redo command, Edit menu, 4-10
regular expression characters (table), 4-15 to 4-16
Regular Expression option, Find command
  Array and String Display windows, 7-6
  Source, Interactive Execution, and Standard Input/Output windows, 4-14
Reload DLLs On Each Run option, 3-63
Remove File command, Edit menu, 3-8
Replace command, Edit menu
  button bar (figure), 4-17
  Find Next button, 4-17
  keyboard commands for bypassing dialog box (table), 4-18
  Replace button bar (figure), 4-17
  Replace All button, 4-17
  Return button, 4-18
  Stop button, 4-18
Require function prototypes option, 3-57 to 3-58, 4-4
Require return values for non-void functions option, 3-58
resdir configuration option, 1-5

Reset Indices command, Options menu
description, 7-9
displaying single-dimensional arrays, 7-3
specifying index for string array, 7-4
specifying plane and dimensions for multi-dimensional arrays, 7-2
Reset Indices dialog box, 7-3
Resolve All Excluded Lines command, Edit menu, 4-12
Retrace Pointer Chain command, View menu, 6-12
return value controls, 5-4 to 5-5
return values, requiring for non-void functions, 3-58
ring control parameters, specifying, 5-6
RS-232 command, Library menu, 3-43
RS-232 Library, 3-43
Run Function Panel command, Code menu, 5-9
Run Interactive Statements command, Run menu, 4-28 to 4-29
Run menu
Array and String Display windows, 7-9
Project window
Break at First Statement command, 3-31
Breakpoints command, 3-31
Continue command, 3-31
Execute command, 3-32
illustration, 3-30
Run Project command, 3-31
Select External Process command, 3-32
Terminate Execution command, 3-31
Source, Interactive Execution, and Standard Input/Output windows
Activate Panels When Resuming option, 4-32 to 4-33
Add Watch Expression command, 4-33
Break at First Statement command, 4-27, 4-30
Breakpoints command, 4-27, 4-31 to 4-32
Close Libraries command, 4-30
Continue command, 4-29
Down Call Stack command, 4-33
Dynamic Memory command, 4-33
Finish Function command, 4-30
Go to Cursor command, 4-29
illustration, 4-26
Run Interactive Statements command, 4-28 to 4-29
Run Project command, 4-28 to 4-29
Stack Trace command, 4-33
Step Into command, 4-30
Step Over command, 4-29
Terminate Execution command, 4-30
Toggle Breakpoint command, 4-27, 4-30
Up Call Stack command, 4-33
View Variable Value command, 4-33, 6-1, 7-4
Variables and Watch windows, 6-14
Run Options command, Options menu, 3-62 to 3-63
Break on library errors option, 3-62
Check disk dates before each run, 3-62
Debugging level, 3-62
Hide windows, 3-62
Maximum stack size (bytes), 3-62
Reload DLLs on each run, 3-63
Save changes before running, 3-62
Unload DLLs after each run, 3-63
Run Project command, Run menu
Project window, 3-31
Source, Interactive Execution, and Standard Input/Output windows, 4-28 to 4-29
-run startup option (table), 1-1
-run_then_exit startup option (table), 1-1

run-time error reporting
Project window, 3-31
Source, Interactive Execution, and Standard Input/Output windows, 4-29
Run-Time Errors command, Window menu, 3-54

## S

Save command, File menu
Project window, 3-5
Source, Interactive Execution, and Standard Input/Output windows, 4-8
Save All command, File menu
Array and String Display windows, 7-6
Function Panel windows, 5-8
Project window, 3-5
Source, Interactive Execution, and Standard Input/Output windows, 4-8
Variables and Watch windows, 6-6
Save As command, File menu
Project window, 3-5
Source, Interactive Execution, and Standard Input/Output windows, 4-8
Save changes before running option, 3-62
Save Copy As command, File menu, 4-8
sdk directory (table), 1-5
Search for Help On command, Help menu, 4-42
Select ActiveX Automation Server dialog box
Automation Servers list box, 3-47
Browse button, 3-47
Done button, 3-47
illustration, 3-47
Open button, 3-47
Select All command, Edit menu, 4-11
Select Attribute Constant command, Code menu
attribute control constants, 5-13 to 5-14
value control constants, 5-14 to 5-15
Select Attribute Constant Dialog Box, 5-13 to 5-15
Select Attribute Values Dialog Box, 5-15
Select External Process command, Run menu, 3-32
Select Function Panel dialog box, 3-40 to 3-41
Alphabetize command, 3-40
Flatten checkbox, 3-40, 5-2
Function Names command, 3-40
Help button, 3-41
illustration, 3-40
New Window command, 3-40
Select Target File dialog box, 3-51
Browse button, 3-51
Cancel button, 3-51
illustration, 3-51
OK button, 3-51
Select Target .fp File, 3-51
Target .c File, 3-51
Target .h File, 3-51
Select UIR Constant command, Code menu, 5-11 to 5-12
Select UIR Constant Dialog Box, 5-12
Select Variable command, Code menu, 5-15 to 5-18
Select Variable or Expression Dialog Box, 5-16 to 5-18
data type compatibility, 5-17 to 5-18
Data Type list box, 5-16
Data Type of Control display, 5-16
illustration, 5-16
items included in list box, 5-17
Show Project Variables option, 5-16
sorting of list box entries, 5-18
Variable or Expression list box, 5-16
Selected Text Only option, Find command, 4-16
separators, adding and positioning on toolbar, 4-3
Set Target File command, Code menu, 5-18

Show Build Error window for warnings option, 3-59
Show Full Dates command, View menu, 3-9
Show Full Path Names command, View menu, 3-8
Show Info command, Edit Instrument dialog box, 3-39
signed/unsigned pointer mismatch warning, enabling, 3-58
single-dimensional array, displaying in Array Display window (figure), 7-2
skeleton code, 2-6
Sleep policy when not running program option, 3-63
slide controls
  definition, 5-5
  specifying parameters, 5-5
Sort By Date command, View menu, 3-9
Sort By File Extension command, View menu, 3-9
Sort By Name command, View menu, 3-9
Sort By Pathname command, View menu, 3-9
source files
  debugging, 2-6
  generating DLL glue source, 4-40 to 4-41
  listed in Window menu, 3-56
  required in project file list, 3-1
Source option, Add Files to Project command, 3-7
Source window
  Build menu, 4-23 to 4-25
  context menus, 4-3
  Edit menu, 4-9 to 4-18
  File menu, 4-7 to 4-9
  Instrument menu, 4-34
  keyboard commands (figure), A-1 to A-2
  Library menu, 4-34
  notification of external modification, 4-3
  opening
    with New command, 3-4
    with Open command, 3-5
  Options menu, 4-36 to 4-42
  purpose and use, 2-4, 4-1
  Run menu, 4-26 to 4-33
  selecting text, 4-5 to 4-7
  subwindows, 4-5
  Tools menu, 4-34 to 4-36
  View menu, 4-19 to 4-23
  Window menu, 4-36
space character, difference from NUL byte (note), 6-1, 7-1
stack size, setting, 3-62
Stack Trace command, Run menu, 4-33
  Down Call Stack, 4-33
  Up Call Stack, 4-33
standalone executables, creating and distributing
  Create Distribution Kit command, 3-24 to 3-30
  Create Standalone Executable dialog box, 3-14 to 3-15
Standard Input/Output command, Window menu, 3-56
Standard Input/Output window
  bringing to front whenever modified, 3-64
  Build menu, 4-23 to 4-25
  clearing, 4-5
  Edit menu, 4-9 to 4-18
  File menu, 4-7 to 4-9
  Instrument menu, 4-34
  Library menu, 4-34
  Options menu, 4-36 to 4-42
  purpose and use, 2-5, 4-5
  Run menu, 4-26 to 4-33
  selecting text, 4-5 to 4-7
  specifying maximum number of lines for, 3-64, 4-5
  subwindows, 4-5
  Tools menu, 4-34 to 4-36
  View menu, 4-19 to 4-23
  Window menu, 4-36

standard libraries. *See also* specific libraries.
list of libraries, 2-2
specifying optional libraries to be loaded, 3-66
startup options for LabWindows/CVI (table), 1-1 to 1-2
static library, creating, 3-18 to 3-19
status dialog, displaying, 3-59
Step Into command, Run menu, 4-30
Step Over command, Run menu, 4-29
Stop on first file with errors option, 3-59
String Display command, View menu, 6-13, 7-4
String Display window
displayed in View menu, 3-54
Edit menu, 7-7 to 7-8
File menu, 7-5 to 7-6
Format menu, 7-8
illustration, 7-4
multi-dimensional strings, 7-4
Options menu, 7-9 to 7-10
purpose and use, 2-4, 7-4
Run menu, 7-9
Window menu, 7-9
structures
child structure, 6-3
child structure pointer in chain (figure), 6-12
Follow Pointer Chain command, 6-11 to 6-12
parent pointer to structure, 6-3
parent structure pointer in chain (figure), 6-11
pointer-linked structures, 6-11
replacing, 6-11 to 6-12
Retrace Pointer Chain command, 6-12
subwindows, in Source, Interactive Execution, and Standard Input/Output windows, 4-5
Syntax Coloring option, Options menu, 4-39
system libraries, installing
Microsoft Windows, 3-46
UNIX, 3-46

## T

Tabs option, Editor Preferences command, 4-38
Tag Scope command, View menu, 4-20
tagged lines
Clear Tags command, 4-20
Next Tag command, 4-20
Previous Tag command, 4-20
Tag Scope command, 4-20
Toggle Tag command, 4-20
Target command, Build menu, 3-11
TCP command, Library menu, 3-44
TCP Library, 3-44
technical support, B-1 to B-2
telephone and fax support numbers, B-2
Terminate Execution command, Run menu
Project window, 3-31
Source, Interactive Execution, and Standard Input/Output windows, 4-30
text, selecting
Character Select mode, 4-6
Column Select mode, 4-7
Line Select mode, 4-6
Tile Windows command, Window menu, 3-53
time option, DSTRules, 1-6
timer option, useDefaultTimer, 1-6
tmpdir configuration option, 1-5
Toggle Breakpoint command, Run menu, 4-27, 4-30
Toggle Control Style command, Options menu, 5-22
Toggle Exclusion command, Edit menu, 4-4, 4-12
Toggle Tag command, View menu, 4-20

tokens
Syntax Coloring option, Options menu, 4-39
User Defined Tokens for Coloring command, Options menu, 4-39
Toolbar command
Options menu
Function Panel windows, 5-22
Source, Interactive Execution, and Standard Input/Output windows, 4-38
View menu
Function Panel windows, 5-19
Source, Interactive Execution, and Standard Input/Output windows, 4-20
toolbars, 4-1 to 4-3
Customize Toolbar Dialog Box (illustration), 4-2
displaying names of button or icons, 4-1
function panels, 5-3
modifying, 4-2 to 4-3
positioning buttons and separators, 4-2 to 4-3
removing items, 4-3
Tools menu
Project window, 3-46 to 3-52
Create ActiveX Automation Controller command, 3-46 to 3-52
Create IVI Instrument Driver command, 3-52
installing user-defined entries, 3-52
Source, Interactive Execution, and Standard Input/Output windows, 4-34 to 4-36
Create ActiveX Automation Controller command, 4-34
Create IVI Instrument Driver command, 4-35
Edit Function Panel command, 4-36
Edit Function Tree command, 4-35
Edit Instrument Attributes command, 4-35
Update ActiveX Automation Controller command, 4-34 to 4-35
Track include file dependencies option, 3-59
Translate LW DOS program command, Options menu, 4-39
Type option, Find command, 6-8

## U

.uir files. *See* user interface resource (.uir) files.
Undo command, Edit menu, 4-10
Undo option, Editor Preferences command, 4-10, 4-37
UNIX operating system
configuration options, 1-3
cfgdir, 1-4
cvidir, 1-4 to 1-5
setting, 1-3
tmpdir, 1-5
date and time option: DSTRules, 1-6
display, mouse, and keyboard options, 1-8 to 1-9
activate, 1-8
useDefaultColors, 1-9
useMetaKey, 1-9
warpMouseOverDialogBoxes, 1-9
font options
appFont, 1-8
dialogFont, 1-8
editorFont, 1-8
menuFont, 1-8
messageBoxFont, 1-8
installing system libraries, 3-46
Unload command, Instrument menu, 3-38, 3-45
Unload DLLs after each run, 3-63
unloading instrument drivers, 3-34 to 3-36
unreachable code warning, enabling, 3-59

Up Call Stack command, Run menu, 4-33
Update ActiveX Automation Controller command, Tools menu, 4-34 to 4-35
Update Program Files from Disk command, Build menu, 3-11
Use host's system standard I/O option, Environment command, 3-64
Use only one Function Panel option, Environment command, 3-64
Use only one Function Panel Window option, Environment command, 3-40
useDefaultColors option, UNIX, 1-9
useDefaultTimer option, 1-6
useMetaKey option, UNIX, 1-9
User Defined Tokens for Coloring command, Options menu, 4-39
User Interface command, Library menu, 3-42
user interface constants, selecting, 5-11 to 5-12
  attribute constants, 5-13 to 5-15
  from .uir files, 5-12
  value constants, 5-14 to 5-15
User Interface Editor window
  moving to with Find UI Object command, 4-23
  opening
    with New command, 3-4
    with Open command, 3-5
  purpose and use, 2-5
User Interface Library
  definition, 3-42
  purpose and use, 2-3
user interface objects, finding, 4-23
User Interface option, Add Files to Project command, 3-7
user interface resource (.uir) files
  displayed in Window menu, 3-55
  optional for project file list, 3-1
user libraries. *See also* libraries.
  dummy .fp files for support libraries, 3-45 to 3-46, 3-66
  installing into Library menu, 3-45 to 3-46
  instrument drivers *vs.*, 3-65
  specifying in Library Options dialog box, 3-65 to 3-66
Utility command, Library menu, 3-45
Utility Library
  definition, 3-45
  functions in instrsup.dll, 3-13

## V

value constants, selecting, 5-14 to 5-15
Value option, Find command, 6-8
Variable Size command, Options menu, 6-15
variables, selecting. *See* Select Variable or Expression Dialog Box.
Variables command, Window menu
  Project window, 3-54
  Variables window, 6-1
Variables window, 6-1 to 6-3
  Edit menu, 6-6 to 6-8
  File menu, 6-5 to 6-6
  Format menu, 6-13
  Function subwindow, 6-2
  Global subwindow, 6-2
  icons associated with variables, 6-2
  illustration, 6-2
  Options menu, 6-14 to 6-15
  purpose and use, 2-4
  Run menu, 6-14
  View menu, 6-10 to 6-13
  viewing, 6-1
  Window menu, 6-14
View menu
  Function Panel windows
    Current Tree command, 5-20
    Error command, 5-19
    Find Function Panel command, 5-20
    First Function Panel Window command, 5-21
    First Panel command, 5-2

Function Panel History command, 5-20
illustration, 5-19
Include File command, 5-20
Last Function Panel Window command, 5-21
Last Panel command, 5-2
Next Function Panel command, 5-20
Next Function Panel Window command, 5-21
Next Panel command, 5-2
Previous Function Panel command, 5-20
Previous Function Panel Window command, 5-21
Previous Panel command, 5-2
Toolbar command, 5-19
Project window, 3-8 to 3-9
illustration, 3-8
No Sorting command, 3-9
Show Full Dates command, 3-9
Show Full Path Names command, 3-8
Sort By Date command, 3-9
Sort By File Extension command, 3-9
Sort By Name command, 3-9
Sort By Pathname command, 3-9
Source, Interactive Execution, and Standard Input/Output windows
Beginning/End of Selection command, 4-20
Build Errors in Next File command, 4-25
Clear Tags command, 4-20
Find Function Panel command, 4-22
Find UI Object command, 4-23
Function Panel History command, 4-21
Function Panel Tree command, 4-21
illustration, 4-19
Line command, 4-20
Line Icons command, 4-19, 4-27
Line Numbers command, 4-19
Next Tag command, 4-20
Previous Tag command, 4-20
Recall Function Panel command, 4-21 to 4-22
Tag Scope command, 4-20
Toggle Tag command, 4-20
Toolbar command, 4-20
Variables and Watch windows
Array Display command, 6-13, 7-1
Close Variable command, 6-3, 6-11
Expand Variable command, 6-3, 6-10 to 6-11
Follow Pointer Chain command, 6-3, 6-11 to 6-12
Go to Definition command, 6-12
Go to Execution Position command, 6-12
illustration, 6-10
Retrace Pointer Chain command, 6-12
String Display command, 6-13, 7-4
View Variable Value command
Code menu, 5-7, 5-19, 6-1, 7-4
Run menu, 4-33, 6-1, 7-4
Virtual Instrument Software Architecture Library. *See* VISA Library.
VISA command, 3-44
VISA Library, 2-3
Visit LabWindows/CVI Web Page command, Help menu, 4-42
Visual Basic, generating include file for, 4-41
VXI command, Library menu, 3-43
VXI*plug&play* instrument driver files, 3-33 to 3-34


## W


Warp mouse over dialog boxes environment option, 3-64

warpMouseOverDialogBoxes option, UNIX, 1-9
Watch command, Window menu
Project window, 3-54
Watch window, 6-4
watch variables/expressions
Add/Edit Watch Expression dialog box, 6-4 to 6-5
applicable only in source code modules (note), 4-27
purpose and use, 4-26 to 4-27
selecting, 6-4 to 6-5
suspending program execution conditionally, 4-28
Watch window, 6-3 to 6-5
activating, 6-3
Add/Edit Watch Expression dialog box, 6-4 to 6-5
Edit menu, 6-9
File menu, 6-5 to 6-6
Format menu, 6-13
illustration, 6-4
purpose and use, 2-4, 6-3
selecting variables and expressions, 6-4 to 6-5
View menu, 6-10 to 6-13
Window menu, 6-14
Whole Word option, Find command
Array and String Display windows, 7-6
Source, Interactive Execution, and Standard Input/Output windows, 4-14
Variables window, 6-7
WIN32 macro, 3-60
_WIN32 macro, 3-60
__WIN32__ macro, 3-60
Window Help dialog box, 3-41
Window menu
Array/String Display windows, 3-54, 7-9
Function Panel windows, 3-55, 5-21
Help Editor windows, 3-55
Project window
Build Errors command, 3-54
Cascade Windows command, 3-53
Close All command, 3-53
Function Tree files, 3-55
illustration, 3-53
Interactive Execution command, 3-55
Minimize All command, 3-53
open source files, 3-56
Project command, 3-54
Run-Time Errors command, 3-54
Standard Input/Output command, 3-56
Tile Windows command, 3-53
User Interface Resource files, 3-55
Variables command, 3-54
Watch command, 3-54
Source, Interactive Execution, and Standard Input/Output windows, 4-36
Variables window, 6-1, 6-14
Watch window, 6-3, 6-14
Windows
configuration options
cfgdir (Windows 3.1), 1-4
cvidir, 1-4 to 1-5
setting
Win95/NT environment settings stored in registry, 1-4
Windows 3.1, 1-3
Windows 95/NT, 1-2
tmpdir, 1-5
date and time option: DSTRules, 1-6
debug options (Windows 3.1)
CatchProtectionFaults option, 1-7
DisplayCVIDebugVxDMissingMessage, 1-7
LoadCVIDebugVxD, 1-7
DLLs. *See* Windows DLLs.

font options
DialogFontBold, 1-7
DialogFontName, 1-6
DialogFontSize, 1-7
installing system libraries, 3-46
timer option: useDefaultTimer, 1-6
windows, hiding, 3-62
Windows DLLs. *See also* Create Dynamic Link Library dialog box.
DLL Debugging control, Build menu, 3-19 to 3-21
DLL import library, generating, 4-40
DLL option, Add Files to Project command, 3-7
DLL path option, Add Files to Project command, 3-7
glue code, generating, 4-40 to 4-41
instrusup.dll, 3-12 to 3-13
Reload DLLs On Each Run option, 3-63
source code for creating DLL import library, generating, 4-39 to 4-40
Unload DLLs After Each Run option, 3-63
_WINDOWS macro, 3-60
Windows SDK command, Help menu, 4-42
Wrap option, Find command
Array and String Display windows, 7-6
Source, Interactive Execution, and Standard Input/Output windows, 4-16
Variables window, 6-7

## X

X Client Property Library, 3-44
X Property command, 3-44
.Xdefaults file, setting configuration options, 1-3