Internet Support

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

Bulletin Board Support

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

FaxBack Support

(512) 418-1111

Telephone Support (U.S.)

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

International Offices

Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico 95 800 010 0793,
Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085,
Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

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

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

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

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

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

Copyright

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

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

Operating the File Select Pop-Up Panel .......................................................... 1-23
Operating Graph Pop-Up Panels ................................................................. 1-24
Using Fonts .................................................................................................. 1-25
Metafonts That Use Typefaces Native To Each Platform ......................... 1-25
Fonts That Use Typefaces Native To Each Platform ................................. 1-25
Metafonts That Use Typefaces Installed by LabWindows/CVI ............... 1-25

Chapter 2
User Interface Editor Reference....................................................................... 2-1
User Interface Editor Overview ................................................................... 2-1
Using the User Interface Editor Popup Menus: ....................................... 2-2
Code Builder Overview............................................................................. 2-2
User Interface Editor Menus ..................................................................... 2-3
File Menu .................................................................................................. 2-3
New, Open, Save, and Exit LabWindows/CVI Commands ....................... 2-4
Save As and Close Commands ................................................................ 2-4
Save Copy As .......................................................................................... 2-4
Save All ................................................................................................... 2-4
Add File to Project .................................................................................. 2-4
Read Only ............................................................................................... 2-4
Print ....................................................................................................... 2-4
Edit Menu ............................................................................................... 2-5
Undo and Redo ...................................................................................... 2-5
Cut and Copy .......................................................................................... 2-6
Paste ...................................................................................................... 2-6
Delete ..................................................................................................... 2-6
Copy Panel and Cut Panel ...................................................................... 2-6
Menu Bars ............................................................................................... 2-7
Panel ....................................................................................................... 2-9
Control ................................................................................................... 2-11
Tab Order ............................................................................................... 2-14
Set Default Font .................................................................................... 2-15
Apply Default Font ............................................................................... 2-15
Control Style ......................................................................................... 2-16
Create Menu .......................................................................................... 2-16
Panel ....................................................................................................... 2-16
Menu Bar ............................................................................................... 2-16
Controls ................................................................................................. 2-16
View Menu ............................................................................................ 2-17
Find UIR Objects.................................................................................... 2-17
Show/Hide Panels .................................................................................. 2-19
Preview User Interface Header File ....................................................... 2-19
Arrange Menu ........................................................................................ 2-20
Alignment .............................................................................................. 2-20
Align Horizontal Centers ....................................................................... 2-21
Distribution ........................................................................................... 2-21
Chapter 3
Programming with the User Interface Library ................................................... 3-1

Developing and Running a Program ................................................................. 3-1
Creating a Graphical User Interface .................................................................. 3-2
If you design your GUI in the User Interface Editor ....................................... 3-2
If you design your GUI programmatically ...................................................... 3-2
Assigning Constant Names in the User Interface Editor ................................. 3-2
Controlling a Graphical User Interface ............................................................ 3-3
User Interface Events ....................................................................................... 3-4
Using Callback Functions to Respond to User Interface Events ................. 3-5
Using GetUserEvent to Respond to User Interface Events .......................... 3-8
Programming with Panels ............................................................................. 3-9
Panel Functions ............................................................................................... 3-9
Role of Child Panels...................................................................................... 3-11
Chapter 4
User Interface Library Reference

User Interface Library Overview ................................................................. 4-1
User Interface Function Panels ............................................................... 4-1
Hard Copy Output ..................................................................................... 4-9
   Compatible Printers .............................................................................. 4-9
   Obtaining Hard Copy Output ............................................................. 4-9
   Configuring Your System for Hard Copy Output .............................. 4-9
RGB Color Values .................................................................................. 4-10
Fonts ....................................................................................................... 4-10
Shortcut Keys ......................................................................................... 4-10
Plot Array Data Types .......................................................................... 4-11
Include Files ......................................................................................... 4-12
Reporting Errors ................................................................................... 4-12
User Interface Library Function Reference ............................................. 4-12
   AllocBitmapData ............................................................................... 4-12
   AllocImageBits .................................................................................. 4-13
   CanvasClear ...................................................................................... 4-15
   CanvasDefaultPen .............................................................................. 4-16
   CanvasDimRect .................................................................................. 4-17
   CanvasDrawArc .................................................................................. 4-18
   CanvasDrawBitmap .......................................................................... 4-20
   CanvasDrawLine ............................................................................... 4-21
   CanvasDrawLineTo .......................................................................... 4-23
   CanvasDrawOval ............................................................................... 4-23
   CanvasDrawPoint .............................................................................. 4-25
   CanvasDrawPoly ............................................................................... 4-26
   CanvasDrawRect ............................................................................... 4-27
   CanvasDrawRoundedRect ................................................................ 4-28
   CanvasDrawText ............................................................................... 4-29
   CanvasDrawTextAtPoint .................................................................... 4-31
   CanvasEndBatchDraw ....................................................................... 4-33
   CanvasGetClipRect .......................................................................... 4-34
   CanvasGetPenPosition ..................................................................... 4-35
   CanvasGetPixel ................................................................................. 4-36
   CanvasGetPixels .............................................................................. 4-37
   CanvasInvertRect ............................................................................. 4-38
   CanvasScroll ..................................................................................... 4-39
   CanvasSetClipRect ........................................................................... 4-40
   CanvasSetPenPosition ...................................................................... 4-41
   CanvasStartBatchDraw ..................................................................... 4-42
   CanvasUpdate .................................................................................... 4-43
   CheckListItems .................................................................................. 4-44
   ClearAxisItems ................................................................................. 4-45
ClearListCtrl .............................................................. 4-46
ClearStripChart ......................................................... 4-46
ClipboardGetBitmap .................................................. 4-47
ClipboardGetText ...................................................... 4-48
ClipboardPutBitmap .................................................. 4-48
ClipboardPutText ...................................................... 4-49
ConfigurePrinter ...................................................... 4-50
ConfirmPopup ......................................................... 4-50
CreateMetaFont ....................................................... 4-51
DefaultCtrl ............................................................. 4-53
DefaultPanel ........................................................... 4-53
DeleteAxisItem ......................................................... 4-54
DeleteGraphPlot ...................................................... 4-55
DeleteImage ............................................................ 4-57
DeleteListItem ......................................................... 4-57
DeleteTextBoxLine .................................................... 4-58
DirSelectPopup ...................................................... 4-59
DiscardBitmap ...................................................... 4-60
DiscardCtrl ............................................................. 4-60
DiscardMenu .......................................................... 4-61
DiscardMenuBar ..................................................... 4-61
DiscardMenuItem ..................................................... 4-62
DiscardPanel .......................................................... 4-62
DiscardSubMenu ..................................................... 4-63
DisplayImageFile .................................................... 4-64
DisplayPCXFile ...................................................... 4-65
DisplayPanel ........................................................... 4-65
DOSColorToRGB ..................................................... 4-66
DOSCompatWindow ................................................ 4-67
DuplicateCtrl .......................................................... 4-68
DuplicatePanel ....................................................... 4-69
EmptyMenu ............................................................. 4-70
EmptyMenuBar ........................................................ 4-70
FakeKeystroke ........................................................ 4-71
FileSelectPopup ...................................................... 4-71
FontSelectPopup ..................................................... 4-73
GenericMessagePopup ............................................ 4-76
Get3dBorderColors .................................................. 4-78
GetActiveCtrl ........................................................ 4-79
GetActiveGraphCursor ............................................ 4-79
GetActivePanel ...................................................... 4-80
GetAxisItem ............................................................ 4-81
GetAxisItemLabelLength .......................................... 4-82
GetAxisRange ........................................................ 4-83
GetAxisScalingMode ............................................... 4-84
GetBitmapData ........................................................ 4-86
GetBitmapFromFile ...................................................................................... 4-88
GetBitmapInfo............................................................................................... 4-89
GetCtrlAttribute ............................................................................................ 4-90
GetCtrlBitmap ............................................................................................... 4-91
GetCtrlBoundingRect.................................................................................... 4-92
GetCtrlDisplayBitmap................................................................................... 4-93
GetCtrlIndex.................................................................................................. 4-94
GetCtrlVal ..................................................................................................... 4-95
GetCursorAttribute........................................................................................ 4-96
GetCVITaskHandle....................................................................................... 4-97
GetCVIWindowHandle................................................................................. 4-97
GetGlobalMouseState ................................................................................... 4-98
GetGraphCursor ............................................................................................ 4-99
GetGraphCursorIndex ................................................................................... 4-100
GetImageBits................................................................................................. 4-101
GetImageInfo ................................................................................................ 4-104
GetIndexFromValue...................................................................................... 4-105
GetLabelFromIndex ...................................................................................... 4-106
GetLabelLengthFromIndex........................................................................... 4-106
GetListItemImage ......................................................................................... 4-107
GetMenuBarAttribute ................................................................................... 4-108
GetMouseCursor ........................................................................................... 4-109
GetNumAxisItems......................................................................................... 4-109
GetNumCheckedItems .................................................................................. 4-111
GetNumListItems .......................................................................................... 4-111
GetNumTextBoxLines .................................................................................. 4-112
GetPanelAttribute.......................................................................................... 4-113
GetPanelDisplayBitmap ................................................................................ 4-113
GetPanelMenuBar ......................................................................................... 4-115
GetPlotAttribute ............................................................................................ 4-115
GetPrintAttribute........................................................................................... 4-116
GetRelativeMouseState................................................................................. 4-117
GetScreenSize ............................................................................................... 4-118
GetSharedMenuBarEventPanel..................................................................... 4-118
GetSleepPolicy .............................................................................................. 4-119
GetSystemAttribute....................................................................................... 4-119
GetSystemPopupsAttribute ........................................................................... 4-120
GetTextBoxLine............................................................................................ 4-120
GetTextBoxLineLength................................................................................ 4-121
GetTextDisplaySize ...................................................................................... 4-122
GetTraceAttribute ........................................................................................ 4-122
GetUILTimeString ......................................................................................... 4-123
GetUserEvent ................................................................................................ 4-124
GetValueFromIndex...................................................................................... 4-125
GetValueLengthFromIndex .......................................................................... 4-125
GetWaitCursorState ...................................................................................... 4-126
Contents

HidePanel ...................................................................................................... 4-127
InsertAxisItem ............................................................................................ 4-127
InsertListItem ............................................................................................. 4-129
InsertSeparator ........................................................................................... 4-131
InsertTextBoxLine ....................................................................................... 4-131
InstallCtrlCallback .................................................................................... 4-132
InstallMainCallback ................................................................................. 4-134
InstallMenuCallback .................................................................................. 4-135
InstallMenuDimmerCallback ...................................................................... 4-136
InstallPanelCallback .................................................................................. 4-137
InstallPopup ................................................................................................ 4-138
IsListItemChecked ..................................................................................... 4-139
LoadMenuBar ............................................................................................... 4-139
LoadMenuBarEx ........................................................................................... 4-141
LoadPanel .................................................................................................... 4-142
LoadPanelEx ................................................................................................. 4-144
MakeColor .................................................................................................... 4-145
MakePoint .................................................................................................... 4-147
MakeRect ...................................................................................................... 4-147
MessagePopup ............................................................................................. 4-148
MultiFileSelectPopup .................................................................................. 4-149
NewBitmap ................................................................................................... 4-150
NewCtrl ........................................................................................................ 4-152
NewMenu ..................................................................................................... 4-153
NewMenuBar ................................................................................................. 4-154
NewMenuItem ............................................................................................... 4-155
NewPanel ...................................................................................................... 4-157
NewSubMenu ............................................................................................... 4-159
PlotArc .......................................................................................................... 4-159
PlotBitmap ................................................................................................... 4-161
PlotIntensity ............................................................................................... 4-162
PlotLine ......................................................................................................... 4-165
PlotOval ......................................................................................................... 4-166
PlotPoint ....................................................................................................... 4-167
PlotPolygon ................................................................................................. 4-168
PlotRectangle ............................................................................................. 4-170
PlotScaledIntensity ................................................................................... 4-172
PlotStripChart ............................................................................................. 4-175
PlotStripChartPoint .................................................................................... 4-177
PlotText ......................................................................................................... 4-178
PlotWaveform ............................................................................................. 4-180
PlotX ............................................................................................................. 4-182
PlotXY ........................................................................................................... 4-183
PlotY ............................................................................................................. 4-184
PointEqual ................................................................................................. 4-186
PointPinnedToRect ..................................................................................... 4-187
Contents

SetAxisRange ................................................................. 4-223
SetCtrlAttribute ............................................................... 4-225
SetCtrlBitmap ................................................................. 4-226
SetCtrlIndex ................................................................. 4-228
SetCtrlVal ................................................................. 4-228
SetCursorAttribute .......................................................... 4-229
SetFontPopupDefaults ..................................................... 4-230
SetGraphCursor ............................................................. 4-232
SetGraphCursorIndex ....................................................... 4-233
SetIdleEventRate ........................................................... 4-234
SetImageBits ............................................................... 4-235
SetInputMode ................................................................. 4-238
SetListItemImage ............................................................. 4-239
SetMenuBarAttribute ....................................................... 4-240
SetMouseCursor ............................................................. 4-240
SetPanelAttribute ............................................................ 4-241
SetPanelMenuBar ............................................................ 4-242
SetPanelPos ................................................................. 4-242
SetPlotAttribute .............................................................. 4-243
SetPrintAttribute ............................................................ 4-244
SetSleepPolicy ................................................................. 4-244
SetSystemAttribute ........................................................... 4-245
SetSystemPopupsAttribute .................................................. 4-246
SetTraceAttribute ............................................................... 4-246
SetWaitCursor ................................................................. 4-247
SuspendTimerCallbacks ..................................................... 4-248
UnRegisterWinMsgCallback ................................................. 4-248
ValidatePanel ................................................................. 4-249
WaveformGraphPopup ....................................................... 4-250
XGraphPopup ................................................................. 4-251
XYGraphPopup ............................................................... 4-251
YGraphPopup ............................................................... 4-252

Chapter 5
LabWindows/CVI Sample Programs ........................................ 5-1

Example Program Files .................................................... 5-1

Using the Sample Programs .................................................. 5-2

1. io.prj—Standard I/O ....................................................... 5-3
2. callback.prj—Introduction to Callback Functions ............... 5-3
3. events.prj—User Interface Events .................................. 5-3
4. menus.prj—Menus Controlled by Callback Functions ....... 5-3
5. graphs.prj—Graphs ...................................................... 5-3
6. chart.prj—Strip Charts .................................................. 5-3
7. cursors.prj—Graph Cursors .......................................... 5-3
8. popups.prj—Pop-Up Panels ........................................... 5-3
## Contents

9. listbox.prj — Selection Lists ............................................................... 5-4
10. panels.prj — Child Windows .............................................................. 5-4
11. timerctl.prj — Timer Controls .......................................................... 5-4
12. textbox.prj — Text Boxes ................................................................. 5-4
13. picture.prj — Using Picture Controls .................................................. 5-4
14. build.prj — Building a User Interface Programmatically ............... 5-4
15. getusrev.prj — Programming with Event Loops .............................. 5-4
16. keyfiltr.prj — Handling Keyboard Input ....................................... 5-5
17. moustate.prj — Getting the Mouse State ....................................... 5-5
18. listdelx.prj — Colors in List Boxes ............................................... 5-5
19. 2yaxis.prj — Two Y Axes on a Graph ............................................ 5-5
20. intgraph.prj — Intensity Plots ........................................................... 5-5
21. autostrp.prj — Autoscaling the Y-Axis on a Strip Chart ................. 5-5
22. canvas.prj — Canvas Controls .......................................................... 5-6
23. canvsbmk.prj — Canvas Benchmark .................................................. 5-6
24. drawpad.prj — Using Canvas as Drawing Pad .................................. 5-6
25. piedemo.prj — Pie Chart ................................................................. 5-6
26. imagedit.prj — Changing Image Colors .......................................... 5-6
27. clipbord.prj — Using System Clipboard ........................................... 5-6

## Appendix A

Error Conditions .......................................................................................... A-1

## Appendix B

Customer Communication ............................................................................ B-1

## Glossary

...................................................................................................................... Glossary-1

## Index

...................................................................................................................... Index-1

## Figures

Figure 1-1. A Typical LabWindows/CVI Graphical User Interface ............ 1-1
Figure 1-2. The Out Of Range Dialog Box ................................................ 1-4
Figure 1-3. A Child Panel Within a Parent Panel ...................................... 1-5
Figure 1-4. A Menu Bar and Pull-Down Menu ......................................... 1-6
Figure 1-5. A Pull-Down Menu with a Sub-Menu ..................................... 1-6
Figure 1-6. Numeric Controls ................................................................. 1-8
Figure 1-7. A String Control ................................................................. 1-9
Figure 1-8. A Text Message ...................................................................... 1-10
Figure 1-9. A Text Box Control .............................................................. 1-10
Figure 1-10. The Command Button Controls ......................................... 1-11
Figure 1-11. The Toggle Button Controls ............................................... 1-11
Figure 1-12. The LED Controls .............................................................. 1-12
Figure 1-13. The Binary Switch Controls ............................................... 1-12
Figure 1-14. Ring Controls ...................................................................... 1-13
Figure 1-15. A Ring Control in Pop-Up Format ................................................................. 1-13
Figure 1-16. A Selection List Control in Check Mode ....................................................... 1-14
Figure 1-17. GUI Decorations ............................................................................................ 1-16
Figure 1-18. A Graph Control ............................................................................................ 1-16
Figure 1-19. A Strip Chart Control ..................................................................................... 1-20
Figure 1-20. The Timer Control ......................................................................................... 1-21
Figure 1-21. A Message Pop-Up Panel ............................................................................. 1-22
Figure 1-22. A Generic Message Pop-Up Panel ............................................................... 1-22
Figure 1-23. A Prompt Pop-Up Panel ............................................................................... 1-22
Figure 1-24. A Confirm Pop-Up Panel ............................................................................... 1-23
Figure 1-25. A File Select Pop-Up Panel under Windows 3.1 ............................................ 1-23
Figure 1-26. A File Select Pop-Up Panel under Windows 95 ........................................... 1-24
Figure 1-27. A Graph Pop-Up Panel ................................................................................. 1-24
Figure 2-1. A User Interface Editor Window ...................................................................... 2-1
Figure 2-2. The File Menu ................................................................................................. 2-3
Figure 2-3. The Edit Menu ............................................................................................... 2-5
Figure 2-4. The Menu Bar List Dialog Box ....................................................................... 2-7
Figure 2-5. The Edit Menu Bar Dialog Box ...................................................................... 2-8
Figure 2-6. Source Code Connection ............................................................................... 2-9
Figure 2-7. Panel Attributes ............................................................................................. 2-10
Figure 2-8. Quick Edit Window ....................................................................................... 2-10
Figure 2-9. Source Code Connection ............................................................................... 2-11
Figure 2-10. Control Settings for a Numeric Control ....................................................... 2-12
Figure 2-11. The Edit Label/Value Pairs Dialog Box ....................................................... 2-12
Figure 2-12. Control Appearance for a Numeric Control ................................................. 2-13
Figure 2-13. Label Appearance for a Numeric Control ..................................................... 2-13
Figure 2-14. Quick Edit Window ..................................................................................... 2-14
Figure 2-15. The Edit Tab Order Dialog Box .................................................................. 2-15
Figure 2-16. The Create Menu ....................................................................................... 2-16
Figure 2-17. The View Menu ........................................................................................... 2-17
Figure 2-18. Find UIR Objects Dialog Box ...................................................................... 2-17
Figure 2-19. Find UIR Objects Dialog Box After a Search Executes ............................... 2-18
Figure 2-20. The Show/Hide Panel Sub-Menu ................................................................. 2-19
Figure 2-21. The Arrange Menu ...................................................................................... 2-20
Figure 2-22. The Alignment Menu .................................................................................. 2-20
Figure 2-23. The Distribution Submenu ......................................................................... 2-21
Figure 2-24. The Code Menu .......................................................................................... 2-23
Figure 2-25. The Set Target File Dialog Box ................................................................. 2-23
Figure 2-26. The Generate Menu .................................................................................... 2-24
Figure 2-27. The Generate Code Dialog Box ................................................................. 2-24
Figure 2-28. The Generate All Code Dialog Box ............................................................. 2-25
Figure 2-29. The Generate Main Function Dialog Box .................................................... 2-26
Figure 2-30. The View Menu .......................................................................................... 2-29
Figure 2-31. The Preferences Menu ............................................................................... 2-29
Figure 2-32. The Options Menu ..................................................................................... 2-31
Figure 2-33. The User Interface Preferences Dialog Box .............................................. 2-32
Figure 3-1. The Callback Function Concept ..................................................................... 3-5
Figure 3-2. The Event Loop Concept .............................................................................. 3-8
Figure 3-3. The Geometric Attributes of a Panel ............................................................ 3-18
Contents

Tables

Table 1-1. Keys For Selecting Cursors ................................................................. 1-17
Table 1-2. Image Formats ..................................................................................... 1-20

Table 3-1. User Interface Events ......................................................................... 3-4
Table 3-2. Panel Attributes ................................................................................. 3-12
Table 3-3. Common Color Values ....................................................................... 3-17
Table 3-4. Values and Cursor Styles for ATTR_MOUSE_CURSOR ....................... 3-19
Table 3-5. Font Values ......................................................................................... 3-20
Table 3-6. Menu and Menu Item Attributes ........................................................ 3-24
Table 3-7. Key Modifiers and Virtual Keys .......................................................... 3-26
Table 3-8. Constants for Masking Three Bit Fields in GetMenuBarAttribute ........ 3-28
Table 3-9. Control Attributes .............................................................................. 3-32
Table 3-10. Control Styles for ATTR_CTRL_STYLE ........................................... 3-46
Table 3-11. Control Data Types for the ATTR_DATA_TYPE attribute ................. 3-50
Table 3-12. Numeric Formats ............................................................................. 3-51
Table 3-13. ATTR_CHECK_RANGE Values .......................................................... 3-51
Table 3-14. Control Attributes for Canvas Controls ........................................... 3-56
Table 3-15. Values for ATTR_DRAW_POLICY ..................................................... 3-57
Table 3-16. Values for ATTR_OVERLAPPED_POLICY ......................................... 3-58
Table 3-17. Values for ATTR_PEN_MODE ............................................................ 3-58
Table 3-18. Values and Macros for Rect Structures .......................................... 3-60
Table 3-19. Timer Control Attributes ................................................................. 3-64
Table 3-20. Graph and Strip Chart Attributes .................................................... 3-68
Table 3-21. Cursor Styles for ATTR_CROSS_HAIR_STYLE ............................... 3-78
Table 3-22. Styles for ATTR_CURSOR_POINT_STYLE and ATTR_TRACE_POINT_STYLE .................................................. 3-78
Table 3-23. Line Styles for ATTR_LINE_STYLE .................................................. 3-79
Table 3-24. Plot Styles for ATTR_PLOT_STYLE ................................................ 3-80
Table 3-25. Values for ATTR_PLOT_ORIGIN ...................................................... 3-81
Table 3-26. System Attributes ........................................................................... 3-86
Table 3-27. Hard Copy Attributes ..................................................................... 3-89
Table 3-28. Values for ATTR_COLOR_MODE ..................................................... 3-91

Table 4-1. The User Interface Library Function Tree .......................................... 4-2

Table 5-1. Sample Program Files ....................................................................... 5-2
About This Manual

The LabWindows/CVI User Interface Reference Manual contains information about creating and controlling custom user interfaces with the LabWindows/CVI User Interface Library. To use this manual effectively, you need to be familiar with the information in Getting Started with LabWindows/CVI and the LabWindows/CVI User Manual. Also, you should know how to perform basic tasks with LabWindows/CVI and Windows.

You should read Chapter 1, User Interface Concepts, and Chapter 3, Programming with the User Interface Library, before you develop a program with the User Interface Library. You should also examine and execute the example programs outlined in Chapter 5, LabWindows/CVI Sample Programs. These examples illustrate many of the concepts presented in Chapters 1 and 3 and should help you develop your own user interface.

Organization of This Manual

The LabWindows/CVI User Interface Reference Manual is organized as follows.

- Chapter 1, User Interface Concepts, describes building and controlling a graphical user interface in LabWindows/CVI. It explains how your user interface resource (.uir) files and your code files interact, so that you can structure your event-driven programs correctly. In addition, it describes the objects and concepts that comprise a graphical user interface (GUI). This chapter also describes how to operate menu bars, panels, and controls with the keyboard and mouse.

- Chapter 2, User Interface Editor Reference, tells you how to create a GUI interactively. It describes the User Interface Editor and the procedures for creating and editing panels, controls, and menu bars.

- Chapter 3, Programming with the User Interface Library, describes how to use the User Interface Library in the application programs you create.

- Chapter 4, User Interface Library Reference, describes the functions in the LabWindows/CVI User Interface Library. The User Interface Library Function Overview section contains general information about the User Interface Library functions and panels. The User Interface Library Function Reference section contains an alphabetical list of function descriptions.

- Chapter 5, LabWindows/CVI Sample Programs, contains a list of the sample programs in LabWindows/CVI and a brief description of each.

- Appendix A, Error Conditions, lists the meanings associated with the integer error codes that the LabWindows/CVI User Interface library functions return.
About This Manual

• Appendix B, Customer Communication, contains forms you can use to request help from National Instruments or to comment on our products and manuals.

• The Glossary contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

• The Index contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual.

**bold**  
Bold text denotes a parameter, menu item, return value, function panel item, or dialog box button or option.

*italic*  
Italic text denotes emphasis, a cross reference, or an introduction to a key concept.

**bold italic**  
Bold italic text denotes a note, caution, or warning.

**monospace**  
Text in this font denotes text or characters that you should literally enter from the keyboard. Sections of code, programming examples, and syntax examples also appear in this font. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, variables, filenames, and extensions, and for statements and comments taken from program code.

*italic monospace*  
Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.

< >  
Angle brackets enclose the name of a key. A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys, for example, <Ctrl-Alt-Delete>.

»  
The » symbol leads you through nested menu items and dialog box options to a final action. The sequence File » Page Setup » Options » Substitute Fonts directs you to pull down the File menu, select the Page Setup item, select Options, and finally select the Substitute Fonts option from the last dialog box.

paths  
Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in the following path: drivename\dirname\dir2name\myfile

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

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help you if you have problems with them. To make it easy for you to contact us, this manual contains comment and technical support forms for you to complete. These forms are in Appendix B, Customer Communication, at the end of this manual.
Chapter 1
User Interface Concepts

This chapter describes building and controlling a graphical user interface in LabWindows/CVI. It explains how your user interface resource (.uir) files and your code files interact, so that you can structure your event-driven programs correctly. In addition, it describes the objects and concepts that comprise a graphical user interface. This chapter also describes how to operate menu bars, panels, and controls with the keyboard and mouse.

Introduction to the Graphical User Interface

A LabWindows/CVI graphical user interface (GUI) can consist of panels, command buttons, pull-down menus, graphs, strip charts, knobs, meters, and many other controls and indicators. Figure 1-1 shows a typical GUI created with LabWindows/CVI. You can build a GUI in LabWindows/CVI interactively using the User Interface Editor, a drop-and-drag editor with tools for designing, arranging and customizing user interface objects. With the interactive User Interface Editor, you can build an extensive GUI for your program without writing a single line of code. When you are finished designing your GUI in the User Interface Editor, you save the GUI as a User Interface Resource (.uir) file. Chapter 2, User Interface Editor Reference, has detailed information on the User Interface Editor.

In addition to the User Interface Editor, the User Interface Library has functions for creating or altering the appearance of the controls on your GUI during run-time, so you can programmatically add to, change, or build your entire GUI.

![Figure 1-1. A Typical LabWindows/CVI Graphical User Interface](image-url)
User Interface Events

When you design a user interface, you are defining areas on your computer screen, in the form of controls, that can generate events. For example, when you click on a command button, the button generates a user interface event which is then passed to your C program. Actually, LabWindows/CVI controls generate multiple user interface events. For example, a single mouse-click on a command button will pass the following user interface events to your program for processing.

1. **GOT_FOCUS** event—If the command button is not the active control (does not have the input focus), a mouse-click on the button will make it the active control. When a control gets the input focus, a **GOT_FOCUS** event occurs.

2. **LEFT_CLICK** event—When users click with the left mouse button on the command button, a **LEFT_CLICK** event occurs. LabWindows/CVI user interface controls can recognize left, right, single, and double mouse clicks.

3. **COMMIT** event—When the user releases the mouse button, a **COMMIT** event occurs signifying that the user has performed a commit event on the control.

Each control type available in the LabWindows/CVI User Interface Editor displays different types of information and passes different user interface events to your program. The first half of this chapter introduces each control type available in LabWindows/CVI, telling you how to operate each control type and which events can be passed by each control type.

Controlling Your User Interface Using Callbacks or GetUserEvent

Once you have built your GUI in LabWindows/CVI, you must develop a C program to process the events generated from the user interface and control the flow of your program. LabWindows/CVI offers two basic methods for designing your programs: callback functions and event-loops. When you use callback functions, you write individual functions in your program that are called directly by user interface controls. For example, you may have a function in your program called **AcquireData** which is assigned to a button labeled **Acquire**. Whenever the **Acquire** button is clicked, all of the event information generated by the button will be passed directly to the **AcquireData** function where it is processed.

Alternatively, you can use an event-loop to process your user interface commit events in LabWindows/CVI. With an event-loop, all commit events are funneled through a call to the **GetUserEvent** function in your program. The **GetUserEvent** function identifies which controls generated the events and your program processes the events accordingly.

Both callback functions and event-loops have advantages, depending on your application. You can use either method in your program, or combine methods for added flexibility. For further details, read the section **Creating a Graphical User Interface** in Chapter 3 of this manual, *Programming with the User Interface Library*. 
Source Code Connection

To establish the connection between your .uir files and your C source files you must include a header file in your source code. When you create a control in the User Interface Editor, you must assign some information to the control so that it can be properly accessed by your program. Each control is given a constant name in the User Interface Editor. The constant name is used in functions from the User Interface Library in your program to identify the controls on your GUI. For example, you define a numeric readout on your user interface with the constant name READOUT, residing on a panel with the constant name PANEL. To set the value displayed in the readout from your program, you would call the function SetCtrlVal with the name PANEL_READOUT as one of the parameters to specify the control in which to set the value.

In addition to assigning constant names to each control on your GUI, you can also assign callback functions to controls in the User Interface Editor. For example, if you wanted a function called AcquireData to be executed whenever the command button ACQUIRE is clicked, you assign the callback function name AcquireData to the ACQUIRE button in the User Interface Editor.

Whenever you save a .uir file in the User Interface Editor, LabWindows/CVI automatically creates a corresponding header (.h) file with the same base filename. The header file declares all of the constant names and callback functions that you have assigned within the .uir file. By including the header file in your source code, all of the constant names and callback functions associated with your GUI are automatically declared.

Control Modes for Generating Commit Events

Commit events are generated when the user of the GUI actually commits to an operation such as making a menu selection or typing in a number and pressing <Enter>. When you create a control you can assign one of the following control modes, which determines how the control generates commit events.

- Normal
- Indicator
- Hot
- Validate

Normal specifies that the user can operate the control and that it can be changed programmatically.

Indicator specifies that you can change the control programmatically but users cannot generate commit or value changed events. Strip chart and text message controls are always indicators.
Hot is identical to normal except that the control generates a commit event when acted upon by the user. Normally, a hot control generates a commit event when its state is changed. For example, if the user moves a binary switch from off to on, a commit event is generated. The following control types have unique rules on how commit events are generated.

- For hot numeric, string, and text box controls, a commit event is generated when the user presses <Enter> or <Tab> after entering a value into the control, or if the user clicks elsewhere with the mouse after entering a value into the control.

- For hot selection list controls not in check mode, a commit event is generated when the user presses <Enter> while the control is active, or double-clicks on a list item.

- For hot selection list controls in check mode, a commit event is generated when the user presses the spacebar while the control is active, or double-clicks on a list item.

- For hot graph controls, a commit event is generated when the user moves a cursor with the arrow keys or when the user releases the mouse button after moving a cursor.

Validate is identical to hot except that, before the commit event is generated, the program validates all numeric controls on the panel that have their range-checking attribute set to Notify. LabWindows/CVI checks the control value against a predefined range. If it finds an invalid condition, LabWindows/CVI activates the control, causing a notification box like the one shown in Figure 1-2 to display. The validate control cannot generate a commit event until the user enters a new, valid value into all controls that are out of range. This process ensures that all numeric/scalar controls are valid before the commit event is reported to your application program.

![Figure 1-2. The Out Of Range Dialog Box](image)

Using CodeBuilder to Create Source Code for Your GUI

With the LabWindows/CVI CodeBuilder, you can create automatically complete C code that compiles and runs based on a user interface (.uir) file you are creating or editing. By choosing certain options presented to you in the Code menu of the User Interface Editor, you can produce skeleton code. Skeleton code is syntactically and programmatically correct code that compiles...
and runs before you have typed a single line of code. With the CodeBuilder feature, you save the
time of typing in standard code included in every program, eliminate syntax and typing errors,
and maintain an organized source code file with a consistent programming style. Because a
CodeBuilder program compiles and runs immediately, you can develop and test the project you
create, concentrating on one function at a time.

Operating a Graphical User Interface

Before creating your first GUI using LabWindows/CVI, you must understand the graphical
objects that are available and how they operate. You will learn how to operate all of the GUI
controls and indicators in this section.

Using Panels

A panel is a rectangular region of the screen where controls and menu bars reside. Panels
provide the backdrop for many different activities, such as representing the front panel
instrument.

Panels operate like windows. You can minimize panels, resize them, move them, overlap
multiple panels, or contain another. An example of a child panel within a parent panel is
shown in Figure 1-3. The outer panel is called the parent panel; the inner panel is called a child
panel. Child panels can appear within other child panels. You cannot drag a child panel outside
its parent panel. If you shrink a parent panel, a child panel may be partially or completely
hidden in the shrunken panel view.

![Figure 1-3. A Child Panel Within a Parent Panel](image)

To operate a panel from the keyboard:

- Press <Shift-Ctrl> and the up arrow key to move to the parent panel of the current panel.
- Press <Shift-Ctrl> and the down arrow key to move to the child panel of the current panel.
• Press <Shift-Ctrl> and the left or right arrow key to move to rotate between panels at the current level in the panel hierarchy.

• Press <Tab> to move to the next control on a panel.

• Press <Shift-Tab> to move to the previous control on a panel.

To operate a panel with the mouse:

• Click anywhere inside a panel to make it active.

• Click anywhere on a control to make it active.

Using Menu Bars

A menu bar is a mechanism for encapsulating a set of commands. A menu bar appears as a bar at the top of a panel and contains a set of menu titles. A sample menu bar and pull-down menu appear in Figure 1-4.

![Figure 1-4. A Menu Bar and Pull-Down Menu](image)

A menu title ending in an exclamation point is called an immediate action menu and does not have an associated pull-down menu. When the user clicks on an immediate action menu it executes immediately.

A menu title without an exclamation point has a pull-down menu. The menu contains a collection of menu items and submenu titles. Submenu items are displayed to the side of submenu titles. A sample submenu appears in Figure 1-5.

![Figure 1-5. A Pull-Down Menu with a Sub-Menu](image)

To operate a menu bar from the keyboard:

• To display a menu or execute an immediate action menu, press the <Alt> key and the underlined letter of the menu title. When a menu is displayed, its title is highlighted.
• Press the right or left arrow key to display adjacent menus.

• Press the up or down arrow key or the underlined letter of the menu item to select items in a menu. When an item is selected, its label is highlighted.

• Press <Enter> to execute the highlighted item and remove the menu.

• Press <Esc> to remove the menu without executing a menu item.

To operate a menu bar with the mouse:

• To display a menu or execute an immediate menu, click on the menu title.

• To execute an item within a menu, click on the menu item. The menu disappears.

• Click anywhere outside the menu to remove the menu without executing the menu item.

Menu titles and menu items that are dimmed are not selectable.

Using Controls

A control is an object that resides on a panel to accept input from the user and to display information. The User Interface Library supports the following control types.

• Numeric

• String

• Text Message

• Text Box

• Command Button

• Toggle Button

• LED

• Binary Switch

• Ring Control

• List Box

• Decorations

• Graphs and Strip Charts
• Pictures
• Timer Control
• Canvas Control

To make a control ready to accept input, click on the control with the mouse or press <Tab> or <Shift-Tab> to move from control to control. Pressing <Alt> and the underlined letter in the label of the control makes that control ready to accept input, provided that no menu bars are accessible.

**Data Types of Controls**

Every control has a *data type* associated with it. The data type of the control determines the data type of variables used to set and obtain the value of the control. The following list shows LabWindows/CVI control data types.

```c
unsigned char
char
char *
unsigned short int
short int
unsigned long int
long int
float
double
```

Not all data types are valid for each type of control.

**Numeric Controls**

*Numeric controls* are used to input or display numeric values. A typical use of these controls might be to input or display a voltage value. Numeric controls have many different graphical representations, including knobs, meters, thermometers, gauges, and dials. The numeric controls are shown in Figure 1-6.

![Numeric Controls](image)

Figure 1-6. Numeric Controls
When a numeric control is not restricted to indicator, you may change its value from the keyboard or with the mouse.

To operate a numeric control from the keyboard:

- Press the left or right arrow key to move the cursor within the digital display of the control.
- Press the up or down arrow key to increase or decrease the value displayed in the control.
- Press <Home> to move the cursor to the beginning of the text.
- Press <End> to move the cursor to the end of the text.
- Press <Shift-Home> to highlight all text to the left of the cursor.
- Press <Shift-End> to highlight all text to the right of the cursor.

To operate a numeric control with the mouse:

- Click on the up/down arrows of the digital display to change the value of the control.
- Click on and drag the needle (for circular controls like knobs) or the slider (for slide controls) to change the value of the control.
- Click on scale labels to set the control to the value of the label.

If the control mode of a numeric control is hot or validate, a commit event is generated when the user presses the up or down arrow keys, or when the user changes the digital display of a numeric control and then presses the <Enter> or <Tab> key or clicks on another object with the mouse.

**String Controls**

*String controls* are used to input or view strings of character values. You can use these controls to input a person’s name. A string control is shown in Figure 1-7.

![Figure 1-7. A String Control](image)

To operate a string control:

- Press the left or right arrow key to move the cursor around in the string control.
• Press <Home> to move the cursor to the beginning of the text.

• Press <End> to move the cursor to the end of the text.

If the control mode of the string is hot or validate, a commit event is generated when the user changes the string and then presses the <Enter> or <Tab> key or clicks on another object with the mouse.

Text Messages

Text messages are indicator controls used to programmatically display strings of text. You cannot operate them using the keyboard, but you can assign callback functions to them so that they respond to mouse click events. A text message control is shown in Figure 1-8.

![Warning: Temperature Too High!](image)

Figure 1-8. A Text Message

Text Box Controls

Text box controls are large string controls that feature line, word, and character wrap modes and scroll bars to facilitate displaying large amounts of text. As with the string control, you can interactively enter text into text boxes and you can control the text box programmatically.

A sample text box appears in Figure 1-9.

![Text Box Control](image)

Figure 1-9. A Text Box Control

When entering text into a textbox:

• Press <Ctrl-Tab> to insert four spaces.

• Press <Ctrl-Enter> to start a new line.

• Press <Home> or <End> to move to the beginning or end of the current line.

• Press <Page Up> or <Page Down> to scroll the text box up or down one page.
Text can be selected by holding down <Shift> and pressing the arrow keys, <Home>, <End>, <Page Up>, or <Page Down>.

If the control mode of the text box is hot or validate, a commit event is generated when the user changes the value of the text box and then presses the <Enter> or <Tab> key or clicks on another object with the mouse.

Command Button Controls

*Command buttons* are used to trigger an action, which is indicated by the label on the button. The command buttons are shown in Figure 1-10.

![Square Command Button](image)

**Figure 1-10. The Command Button Controls**

To operate a command button from the keyboard, press <Enter> or <space> to activate the button. If the control mode of the button is hot or validate, a commit event will be generated when the user presses the <Enter> or <space> key. The button changes appearance momentarily to indicate that it was selected.

To operate a push button with the mouse, click on the button. The button remains depressed until the user releases the mouse or moves it off the button. If the control mode of the command button is hot or validate, a commit event is generated when the user clicks and releases the mouse over the command button area. If the user releases the mouse outside of the button area, a commit event is not generated.

Toggle Button Controls

*Toggle buttons* allow you to select between two different states. A toggle button has two positions: pressed or unpressed. When the button is pressed its value is 1, and when it is unpressed its value is 0. The toggle buttons are shown in Figure 1-11.

![Round Button](image)

**Figure 1-11. The Toggle Button Controls**
To operate a toggle button from the keyboard:

- Press the up arrow key or <Home> to press the toggle button in.
- Press the down arrow key or <End> to pop the toggle button out.
- Press <space> or <Enter> to change the state of the button.

To operate a toggle button with the mouse, click on the toggle button to change its state.

If the control mode of the toggle button is hot or validate, a commit event is generated when the button is active and the user changes its state.

**LED Controls**

*LEDs* (Light Emitting Diodes) indicate an on/off state. When the LED is on, its value is 1 and it displays its ON color. When the LED is off, its value is 0 and it displays its OFF color. The LED controls are shown in Figure 1-12.

![Round LED](Round LED)

Figure 1-12. The LED Controls

LED controls operate like toggle button controls.

**Binary Switch Controls**

*Binary switches*, like toggle buttons, allow you to select between two states: on or off. You can also associate a value with each state of a binary switch. The binary switches are shown in Figure 1-13.

![Vertical Switch](Vertical Switch)

Figure 1-13. The Binary Switch Controls

Binary switch controls operate like toggle button controls.
Ring Controls

You use ring controls to select from a group of items. Many of the ring controls look like numeric controls, but ring controls have a finite set of label/value pairs. The ring controls are shown in Figure 1-14.

![Ring Controls](image)

Figure 1-14. Ring Controls

To operate a ring control from the keyboard:

- Press the up arrow key to select the previous ring control item.
- Press the down arrow key to select the next ring control item.

Ring controls with arrows can also be operated in pop-up format, in which a linear list of all the ring items is displayed at once. A sample ring control in pop-up format is shown in Figure 1-15.

![Ring Control in Pop-Up Format](image)

Figure 1-15. A Ring Control in Pop-Up Format

To operate a ring control pop-up with the keyboard:

- Press <space> to display the ring control pop-up.
- Press the up and down arrow keys to highlight particular items.
- Press <Enter> to select the highlighted item. The pop-up control disappears and the ring control is updated to match the new selection.
• Press <Esc> to remove the pop-up without changing the selected item.

To operate a ring with the mouse:

• If the ring control has arrows, click on the arrows to select the ring control items.

To operate a ring control pop-up with the mouse:

• Click on the ring control to display the ring pop-up.

• Click on an item to select it. The pop-up disappears and the ring control is updated to match the new selection.

• When the pop-up exceeds the size of the screen, hold the mouse button down on the top item to scroll up.

• When the pop-up exceeds the size of the screen, hold the mouse button down on the bottom item to scroll down.

• Click on an item to select it. The pop-up disappears and the ring control is updated to match the new selection.

• Click outside the menu to cancel the operation and remove the pop-up.

Note: If the ring pop-up exceeds twice the size of the screen, users see a list box instead of a ring pop-up. See the section List Box Controls in this chapter for instructions on operating list box controls.

If the control mode of a ring control is hot or validate, a commit event is generated when the user changes the ring’s value.

List Box Controls

List box controls are used to select an item from a list. A sample selection list control is shown in Figure 1-16.

Figure 1-16. A Selection List Control in Check Mode
The scroll bar on the right side of the list scrolls the list up and down.

To operate a list box control from the keyboard:

- Press the up arrow key to highlight the previous list item.
- Press the down arrow key to highlight the next list item.
- Press <Home> to scroll to the top of the list.
- Press <End> to scroll to the bottom of the list.
- Press <Page Up> to scroll up one page.
- Press <Page Down> to scroll down one page.
- Use the Quick Type feature to locate a list item. When you type A, Quick Type takes you to the first occurrence of an item beginning with “A.” When you type a complete item name Quick Type takes you to that item. (When you press a key after a delay of at least one second, the Quick Type buffer is flushed.)
- Press the left arrow key to go to the previous item in the Quick Type buffer.
- Press the right arrow key to go to the next item in the Quick Type buffer.
- If the list box is in check mode, press <space> or <Enter> to toggle the check mark of the current item.

To operate a list box control with the mouse:

- Click on an item to highlight it; this action toggles the check mark when in check mode.
- Double-click on an item to select it, when not in check mode.
- Hold the mouse button down on either arrow to scroll through the list.
- Drag the scroll bar marker to a new position.
- Click above the scroll bar marker to scroll up one page.
- Click below the scroll bar marker to scroll down one page.

To generate a commit event when a list box is set to hot or validate mode:

- When the control is set to **Check Mode**, select an item in the list box and press <space> or click on it.
• When the control is not set to **Check Mode**, select an item in the list box and press <Enter> or double-click on it.

**Decorations**

Decorations are used to enhance the visual appeal of the GUI. They do not contain data but they can be assigned callback functions so that they can respond to mouse click events. The decorations are shown in Figure 1-17.

![Figure 1-17. GUI Decorations](image)

**Graph Controls**

*Graph controls* display graphical data as one or more plots. A *plot* consists of a curve, a point, a geometric shape, or a text string. A sample graph control is shown in Figure 1-18.

![Figure 1-18. A Graph Control](image)
Graph controls can have one or more cursors associated with them. With cursors you can select a point or region of the graph for further processing or analysis. If the graph control is hot, cursors will generate commit events. If you want to use cursors, the graph controls must not be in indicator mode.

To operate a graph with cursors using the keyboard, use the keys in the following tables for both free-form and snap-to-point cursors.

### Table 1-1. Keys For Selecting Cursors

<table>
<thead>
<tr>
<th>When you press...</th>
<th>You select...</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Page-Up&gt;</td>
<td>The previous cursor.</td>
</tr>
<tr>
<td>&lt;Page-Down&gt;</td>
<td>The next cursor.</td>
</tr>
<tr>
<td>Left arrow key</td>
<td>Left ten pixels.</td>
</tr>
<tr>
<td>Right arrow key</td>
<td>Right ten pixels.</td>
</tr>
<tr>
<td>Up arrow key</td>
<td>Up ten pixels.</td>
</tr>
<tr>
<td>Down arrow key</td>
<td>Down ten pixels.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-left arrow key</td>
<td>Left one pixel.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-right arrow key</td>
<td>Right one pixel.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-up arrow key</td>
<td>Up one pixel.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-down arrow key</td>
<td>Down one pixel.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-left arrow key</td>
<td>To the left edge of the plot area.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-right arrow key</td>
<td>To the right edge of the plot area.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-up arrow key</td>
<td>To the top edge of the plot area.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-down arrow key</td>
<td>To the bottom edge of the plot area.</td>
</tr>
<tr>
<td>&lt;Home&gt;</td>
<td>To the lower left corner of the plot area.</td>
</tr>
<tr>
<td>&lt;End&gt;</td>
<td>To the upper right corner of the plot area.</td>
</tr>
<tr>
<td>Left arrow key</td>
<td>To the previous point on the current plot.</td>
</tr>
<tr>
<td>Right arrow key</td>
<td>To the next point on the current plot.</td>
</tr>
<tr>
<td>Up arrow key</td>
<td>To the next point on the current plot.</td>
</tr>
<tr>
<td>Down arrow key</td>
<td>To the previous point on the current plot.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-left arrow key</td>
<td>Back 10 points on the current plot.</td>
</tr>
</tbody>
</table>

(continues)
Table 1-1. Keys For Selecting Cursors (Continued)

<table>
<thead>
<tr>
<th>When you press...</th>
<th>You select...</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Shift&gt;-right arrow key</td>
<td>Forward 10 points on the current plot.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-up arrow key</td>
<td>Forward 10 points on the current plot.</td>
</tr>
<tr>
<td>&lt;Shift&gt;-down arrow key</td>
<td>Back 10 points on the current plot.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-left arrow key</td>
<td>Left to the closest point in the X direction on the current plot.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-right arrow key</td>
<td>Right to the closest point in the X direction on the current plot.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-up arrow key</td>
<td>Up to the closest point in the Y direction on the current plot.</td>
</tr>
<tr>
<td>&lt;Ctrl&gt;-down arrow key</td>
<td>Down to the closest point in the Y direction on the current plot.</td>
</tr>
<tr>
<td>&lt;Home&gt;</td>
<td>To the first visible point on the current plot.</td>
</tr>
<tr>
<td>&lt;End&gt;</td>
<td>To the last visible point on the current plot</td>
</tr>
<tr>
<td>&lt;Shift-Page Up&gt;</td>
<td>To the previous plot.</td>
</tr>
<tr>
<td>&lt;Shift-Page Down&gt;</td>
<td>To the next plot.</td>
</tr>
</tbody>
</table>

If the graph is configured as a hot control, an event will be generated whenever the user presses one of the preceding keys.

Operate a graph with cursors using the mouse as follows.

- Drag a cursor to move it. If the cursor is in snap-to-point mode, the cursor tracks the mouse until the user releases the mouse button and then it snaps to the nearest data point. If the cursor is in free-form mode, the cursor tracks the mouse until the user releases the mouse button and then stays at the new position.

- Move the active cursor left and right by dragging the active cursor marker at the top or bottom edge of the plot area. Likewise, you can move the active cursor up and down by dragging the active cursor marker at the left or right edge of the plot area.

If the graph control is configured in hot mode, a commit event is generated whenever the user moves a cursor using the arrow keys or releases the mouse after moving a cursor.

EVENT_VAL_CHANGED events are generated continuously while a user drags a graph cursor.
Zooming and Panning on Graphs

You can use **zooming**—the ability to expand or contract the viewport around a particular point—in graph controls. When you zoom **in**, the logical area contained in the viewport gets smaller, thereby showing the area with more resolution. When you zoom **out**, the viewport shows a wider area. You can also use **panning**, the ability to shift the viewport.

By default, however, zooming and panning are disabled. You must explicitly enable them in the User Interface Editor or programmatically. Also, a graph control must not be in indicator-only mode if zooming and panning are to be used.

To start zooming in on a point, press the <Ctrl> key and left mouse button down over the point. The resolution in the viewport is continuously increased until you release the mouse. (You do not need to keep the <Ctrl> key down.) If you drag the mouse, the zooming continues but does so over the new point under the mouse cursor. The zooming stops when you release the left mouse button or click on the right mouse button.

You zoom out just like you zoom in, except that you use the right mouse button instead of the left mouse button.

To start panning, press the <Ctrl-Shift> keys and the left mouse button over a point on the viewport. Then drag the mouse to another point. The graph viewport is scrolled so that the original point now appears under the new mouse cursor location. You can drag the mouse anywhere on the screen.

To restore the viewport to its original state after zooming or panning, press <Ctrl-Spacebar>.

If you are using auto-scaling in the graph, the auto-scaling must be temporarily disabled while zooming or panning. If any plotting occurs while the end-user is zooming or panning, the zooming or panning is terminated and the new data is shown using auto-scaling.
Strip Chart Controls

Strip chart controls display graphical data in real time. A strip chart consists of one or more traces that are updated simultaneously. A sample strip chart control appears in the following figure.

![Figure 1-19. A Strip Chart Control](image)

Picture Controls

A picture control allows you to place images on panels, such as logos and diagrams. For example, you can use a picture control to place a schematic that instructs the user how to hook up a unit for testing. When you want to place images on command or toggle buttons or on ring controls, LabWindows/CVI frees you from programming the simple Picture control. Separate command and toggle buttons and ring controls exist for pictures. For more information on those picture controls, read the Programming with Picture Controls section of Chapter 3, Programming with the User Interface Library.

The image formats that work with all types of picture controls appear in Table 1-2.

<table>
<thead>
<tr>
<th>Image Format</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCX</td>
<td>Windows and UNIX</td>
</tr>
<tr>
<td>BMP, DIB, RLE, ICO</td>
<td>Windows only</td>
</tr>
<tr>
<td>XWD</td>
<td>UNIX only</td>
</tr>
<tr>
<td>WMF</td>
<td>Windows 95 and NT only</td>
</tr>
</tbody>
</table>
**Timer Controls**

Use *Timer Controls* to trigger actions at specific time intervals. The timer control schedules these actions so that you do not have to. The timer control is shown in Figure 1-20.

![Timer Control](Timer_Control.png)

**Figure 1-20. The Timer Control**

Timer controls repeat a given action at a specified time interval for an indefinite period of time, which makes them useful programming tools when a repeated action is required. You can specify a function to be called at the end of each interval.

LabWindows/CVI has functions to suspend and reset the timer controls.

Timer controls are not visible during program execution. They are only visible in the User Interface Editor. For more information, read the *Programming with Timer Controls* section of Chapter 3, *Programming with the User Interface Library*.

**Canvas Controls**

Use *canvas controls* as an arbitrary drawing surface. You can draw text, shapes, and bitmap images. An offscreen bitmap is maintained so that the appearance of the canvas can be restored when the region is exposed.

If you want to display images that are not rectangular or that have “holes” in them, you can use bitmaps that have a transparent background.

**Using Pop-Up Panels**

A *pop-up panel* is a panel that pops up, accepts user input, and then disappears.

Pop-up panels can be stacked, with each new pop-up panel appearing on top of the previous one. When a pop-up panel is active it appears in the foreground and is the only panel or pop-up you can operate.

The User Interface Library contains a collection of predefined pop-up panels for common operations, such as displaying a multi-line message, prompting the user for input, prompting the user for confirmation, selecting a file, and graphing numerical data.

Specific function calls invoke each predefined pop-up panel. The function displays the pop-up panel and waits for the user to select an action. Then the pop-up disappears and control returns to the program.
Operating the Message Pop-Up Panel

The *message pop-up panel* displays multi-line messages. Use the newline character (`\n`) to start a new line of text. A sample message pop-up panel appears in the following figure.

![Message Pop-Up Panel](image)

Figure 1-21. A Message Pop-Up Panel

Operating the Generic Message Pop-Up Panel

The *generic message pop-up panel* displays a pop-up panel with a message string, a response buffer, and up to three buttons with programmable labels. A sample generic message pop-up panel appears in Figure 1-22.

![Generic Message Pop-Up Panel](image)

Figure 1-22. A Generic Message Pop-Up Panel

Operating the Prompt Pop-Up Panel

The *prompt pop-up panel* requests input from the user. A sample prompt pop-up panel appears in Figure 1-23.

![Prompt Pop-Up Panel](image)

Figure 1-23. A Prompt Pop-Up Panel
Operating the Confirm Pop-Up Panel

The confirm pop-up panel allows the user to confirm an action. A sample confirm pop-up panel appears in Figure 1-24.

![Confirm Pop-Up Panel](image)

Figure 1-24. A Confirm Pop-Up Panel

Operating the File Select Pop-Up Panel

The file select pop-up panel displays a list of files on disk from which the user can select. A file select pop-up panel for Windows 3.1 appears in the following figure.

![File Select Pop-Up Panel under Windows 3.1](image)

Figure 1-25. A File Select Pop-Up Panel under Windows 3.1

A unique feature of this dialog box is the Directories ring in the upper-right corner of the dialog box. When activated, this ring allows users to select from a list of directories that contain previously opened files.
A file select pop-up panel for Windows 95 appears in the following figure.

![Figure 1-26. A File Select Pop-Up Panel under Windows 95](image)

**Operating Graph Pop-Up Panels**

You can choose from four different types of graph pop-up panels to display a graph of numerical data: X-graph pop-up, Y-graph pop-up, XY-graph pop-up, and Waveform graph pop-up. A sample graph pop-up panel appears in the following figure.

![Figure 1-27. A Graph Pop-Up Panel](image)
Using Fonts

Metafonts That Use Typefaces Native To Each Platform

Metafonts contain typeface information, point size, and text styles such as bold, underline, italic, and strikeout. The following metafonts use typefaces that are native to each platform supported by LabWindows/CVI. Each of these metafonts has been specified so that the height and width of the font is as similar as possible across platforms.

VAL_MENU_META_FONT
VAL_MESSAGE_BOX_META_FONT
VAL_DIALOG_META_FONT
VAL_EDITOR_META_FONT
VAL_APP_META_FONT

You can create metafonts from any font on your system with the CreateMetaFont function.

Fonts That Use Typefaces Native To Each Platform

The following fonts use typefaces that are native on both PC and UNIX systems. Fonts contain typeface information only.

VAL_MENU_FONT
VAL_MESSAGE_BOX_FONT
VAL_DIALOG_FONT
VAL_EDITOR_FONT
VAL_APP_FONT

Metafonts That Use Typefaces Installed by LabWindows/CVI

The following metafonts are included in LabWindows/CVI, but use typefaces that are not native to PC or UNIX systems. The typefaces are distributed with LabWindows/CVI and the run-time engine.

VAL_7SEG_META_FONT
VAL_SYSTEM_META_FONT
Chapter 2
User Interface Editor Reference

As explained in Chapter 1, you can create your GUI programmatically using function calls or interactively using the User Interface Editor. This chapter tells you how to create a GUI interactively. It describes the User Interface Editor and the procedures for creating and editing panels, controls, and menu bars.

When you use the User Interface Editor you create and modify user interface resource (.uir) files. Enter the User Interface Editor by selecting New or Open from the File menu and choosing User Interface (*.uir).

User Interface Editor Overview

A User Interface Editor window appears in the following figure.

![Figure 2-1. A User Interface Editor Window](image)

From this window, you can create and edit GUI panels, controls, and menu bars. The menus are described in this chapter in the section User Interface Editor Menus. Use the tool bar beneath the menu bar for high-level editing with the mouse. When you click on a particular tool, the mouse cursor changes to reflect the new editing mode.

You can select, position, and size objects by using the Editing tool.
You can modify text associated with objects by using the Labeling tool.

You can color objects by using the Coloring tool. Clicking the right mouse button displays a color palette from which you can choose a color. Clicking the left mouse button automatically colors the object with the current color of the Coloring tool. Holding down the <Ctrl> key changes the tool to an eyedropper icon. When you click on an object with the eyedropper icon, the current color of the Coloring tool becomes the color of that object. Then you can apply that object’s color to another object.

Use the Operating tool to operate objects. When you are in the operate mode, events display on the right side of the tool bar. These event displays have a built-in delay to give you time to see each event.

Using the User Interface Editor Popup Menus:

You can bring up a pop-up menu by clicking with the right mouse button on the User Interface Editor Window. The type of pop-up menu that appears depends on the surface you click on.

- If you click on the User Interface Editor Window background, a pop-up menu appears containing commands to create a panel or a menu bar.

- If you click on a panel background, a pop-up menu appears with each of the control types you can create.

- If you click on a control, a pop-up menu appears with commands to generate or view the callback function for the control.

Code Builder Overview

With the LabWindows/CVI CodeBuilder, you can create automatically complete C code that compiles and runs based on a user interface (.uir) file you are creating or editing. 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 compiles and runs before you have typed a single line of code. With the CodeBuilder feature, you save the time of typing in standard code included in every program, eliminate syntax and typing errors, and maintain an organized source code file with a consistent programming style. Because a CodeBuilder program compiles and runs immediately, you can develop and test the project you create, concentrating on one function at a time.

When you choose Code » Generate » All Code, LabWindows/CVI places the #include statements, variable declarations, callback function skeletons, and main function in the source code file you specify as the target file. Each function skeleton contains a switch construct with a case statement for every default event you specify. You can set default events for control callback functions and panel callback functions by choosing Code » Preferences. Although
skeleton code runs, you must customize it to implement the actions you want to take place in the case of each event.

When you generate code for a specific control or panel callback function, LabWindows/CVI places the skeleton code for that function in the target file in the same complete format as was done for the Code » Generate » All Code command. However, this code might not run. In order for a project to run, there must be a main function. If you lack the main function or any of the callback functions defined in the .uir file, the code is incomplete.

It is a good idea to use the Code » Generate » All Code option first, to produce a running project from the current state of the .uir file. Then, after adding panels, controls or menu items to the .uir file, use the Generate Panel, Control Callbacks and Menu Callbacks commands to make corresponding additions to the source file.

Also with CodeBuilder, you can take care of program termination for automatically generated code. For a CodeBuilder program to terminate successfully, you must include a call to the QuitUserInterface function. When you choose Code » Generate » All Code, the Generate All Code dialog box prompts you to choose which callback functions terminate the program. You can select one or more callback functions to ensure proper program termination.

User Interface Editor Menus

The User Interface Editor has a menu bar that contains the following options: File, Edit, Create, View, Options, Window, and Help.

File Menu

The File menu is shown in the following figure.

![File Menu](image-url)

Figure 2-2. The File Menu
New, Open, Save, and Exit LabWindows/CVI Commands

The New, Open, Save, and Exit LabWindows/CVI commands in the User Interface Editor menu bar work like New, Open, Save and Exit LabWindows/CVI commands in the Project window. If you require more information, consult Chapter 3, Project Window, in the LabWindows/CVI User Manual.

Save As and Close Commands

The Save As and Close commands work like Save and Close in common Windows applications. It is assumed that you are familiar with these commands.

Save Copy As

The Save Copy As command writes the contents of the active window to disk using a user-specified name, without changing the name of the active window. If you want to append a different extension, type a new extension after the filename. If you want no extension to be appended, enter only a period after the filename.

Save All

The Save All command saves all open files to disk.

Add File to Project

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

Read Only

The Read Only command suppresses the editing capabilities in the current window.

Print

The Print command opens the Print dialog box which allows you to send the entire .uir file or the visible screen area to a printer or a file. The Print dialog box also allows you to set print preferences. The print preferences (attributes) are described in the Generating Hard Copy Output section of Chapter 3, Programming with the User Interface Library.
Edit Menu

Items in this Edit menu are used for editing panels, controls, and menu bars. Figure 2-3 shows the Edit menu.

![Edit Menu](image)

**Figure 2-3. The Edit Menu**

**Note:** *Undo and Redo are enabled when you perform an edit action. The Cut and Copy commands are enabled when you select a control, and Paste is enabled when you place an object in the Clipboard using the Cut or Copy command. If you select an edit command while it is disabled, nothing happens.*

**Undo and Redo**

The **Undo** command reverses your last edit action and the screen returns to its previous state. Edit actions are stored on a stack so that you can undo a series of your edit actions. The stack stores up to 100 edit actions and is set using the Preferences command in the Options menu.

The **Redo** command reverses your last Undo command, restoring the screen to its previous state. Redo is helpful when you use the Undo command to reverse a series of your edit actions and accidentally go too far. The Redo command is enabled only when your previous action was the Undo command. Any action disables the Redo command.

Actions that can be undone and redone are dynamically displayed in the menu, for example, when you move a control the menu presents the option **Undo Move Control**.
Cut and Copy

The Cut and Copy commands put controls in the Clipboard. The Cut command removes the selected control and places it in the Clipboard. The Copy command copies the selected control and places it in the Clipboard, leaving the selected control in its original location. Controls you cut or copy do not accumulate in the Clipboard. Every time you cut or copy a control it replaces whatever was previously stored in the Clipboard.

To use the Cut or Copy commands, follow these steps.

1. Select the control you want to place in the Clipboard by clicking on the control or pressing <Tab> until the control is highlighted. Select multiple controls by dragging the mouse over the controls or <Shift>-clicking on the controls.

2. Select Cut or Copy from the Edit menu.

Paste

The Paste command inserts controls, panels, or text from the Clipboard. You can Paste an object from the Clipboard as many times as you like. Controls or panels remain in the Clipboard until you use Cut, Cut Panel, Copy, or Copy Panel again. The New and Open commands do not erase the Clipboard.

Delete

The Delete command deletes selected controls without placing the controls in the Clipboard. Because Delete does not place controls in Clipboard, you cannot restore the controls using the Paste command.

Copy Panel and Cut Panel

The Copy Panel and Cut Panel commands put an entire panel in the Clipboard. The Cut Panel command removes the selected panel and places it in the Clipboard. The Copy Panel command copies the selected panel and places it in the Clipboard, leaving the selected panel in its original location. Panels you cut or copy do not accumulate in the Clipboard. Every time you cut or copy a panel it replaces whatever was previously stored in the Clipboard.

To use the Cut Panel or Copy Panel commands, follow these steps.

1. Select the panel you want to place in the Clipboard by clicking on the panel or pressing <Shift-Ctrl> and one of the arrow keys (up, down, right, or left) until the panel is activated.

2. Select Cut Panel or Copy Panel from the Edit menu.
Menu Bars

The Menu Bars command brings up the Menu Bar List dialog box shown in the following figure.

![Menu Bar List Dialog Box](image)

Figure 2-4. The Menu Bar List Dialog Box

The list contains all of the menu bars in the resource file, listed by constant prefix. The following list describes the command buttons.

- **Create** brings up a new Edit Menu Bar dialog box (Figure 2-5). After a menu bar is created, it is inserted below the current menu bar in the menu bar list.

- **Edit** brings up the Edit Menu Bar dialog box for the selected menu bar.

- **Cut** deletes the currently highlighted item in the menu bar list and copies it to the menu bar clipboard.

- **Copy** copies the currently highlighted item in the menu bar list to the menu bar clipboard.

- **Paste** inserts the contents of the menu bar clipboard to the menu bar list. The menu bar is inserted above the currently highlighted item in the menu bar list.

- **Done** closes the Menu Bar List dialog box.
The Edit Menu Bar dialog box is shown in the following figure.

![Edit Menu Bar Dialog Box](image)

Figure 2-5. The Edit Menu Bar Dialog Box

The Edit Menu Bar dialog box presents the following options.

- **Menu Bar Constant Prefix** is the resource ID for the menu bar. This resource ID is passed to the `LoadMenuBar` function which loads the panel into memory. The **Menu Bar Constant Prefix** is defined in the `.h` file when you save the `.uir` file. If you do not assign a **Menu Bar Constant Prefix**, the User Interface Editor assigns one for you when you save the `.uir` file.

- **Item** shows the name of the current menu, sub menu, or menu command. If you type two underscores before any letter in the item, the user can select the item by pressing <Alt> and the underlined letter.

- **Constant Name** is appended to the **Menu Bar Constant Prefix** to form the ID for the current item. The ID is used in functions such as `GetMenuBarAttribute` and
SetMenuBarAttribute. The ID is also returned by GetUserEvent when an event is generated by a menu command.

- **Naming a Callback Function** is optional. In this box you can type the name of the function to be called when an event is generated by the current menu item.

- **Modifier Key** and **Shortcut Key** combine to form the hot key for the current item.

- **Dimmed** specifies whether or not the current item is initially dimmed.

- **Checked** specifies whether or not the current item initially has a check mark.

- **Insert New Item** inserts the next item above or below the currently selected item.

- **Insert Separator** inserts a separator (a line) above or below the currently selected item.

- The left hierarchy button, `<<<`, moves the currently selected item up one level in the sub menu hierarchy.

- The right hierarchy button, `>>>`, moves the currently selected item down one level in the sub menu hierarchy.

- **View** displays the current state of the menu bar and pull-down menus.

**Panel**

The Panel command brings up the Edit Panel dialog box. This dialog box has three sections entitled **Source Code Connection**, **Panel Attributes**, and **Quick Edit Window**.

The Source Code Connection section of the Edit Panel dialog box is shown in the following figure.

![Source Code Connection](image)

**Figure 2-6. Source Code Connection**

In the **Constant Name** box you type the resource ID for the panel. You pass this resource ID to the LoadPanel function to load the panel into memory. The **Constant Name** is defined in the .h file when you save the .uir file. If you do not assign a **Constant Name**, the User Interface Editor will assign one when you save the .uir file.
Naming a **Callback Function** is optional. In this box you can type the name of the function to be called when an event is generated on the panel.

The Panel Settings sections of the Edit Panel dialog box appear in the following figure.

![Panel Settings](image)

**Figure 2-7. Panel Attributes**

**Note:** The preceding figure shows settings and attributes available in the Windows version of LabWindows/CVI. The Edit Panel dialog box in the UNIX version of LabWindows/CVI lacks some items because that program cannot update some panel attributes for parent panels.

The Programming with Panels section of Chapter 3, Programming with the User Interface Library, describes panel attributes in detail.

The Quick Edit Window section of the Edit Panel dialog box appears in the following figure.

![Quick Edit Window](image)

**Figure 2-8. Quick Edit Window**
From the Quick Edit Window, you can perform high level edits on the panel. The tools in the tool bar operate like the tools in the main User Interface Editor window. For further information, see the section *User Interface Editor Overview*, earlier in this chapter.

**Control**

The **Control** command brings up the dialog box for editing the selected control. You can also double-click on a control to get this dialog box. The dialog box usually has five sections entitled *Source Code Connection, Control Settings, Control Appearance, Quick Edit Window, and Label Appearance*. The contents of the sections vary slightly depending on the control that is being edited.

The Source Code Connection section of the control dialog box is shown in the following figure.

![Source Code Connection](image)

Figure 2-9. Source Code Connection

The User Interface Editor appends **Constant Name** to the panel resource ID to form the ID for the control. The ID identifies the control in any control-specific functions such as GetCtrlVal and SetCtrlAttribute. The ID will be defined in the .h file when you save the .uir file. If you do not assign a **Constant Name**, the User Interface Editor will assign one for you when you save the .uir file.

Naming a **Callback Function** is optional. In this box you can type the name of the function to be called when an event message is generated on the panel.
The Control Settings section of the dialog box senses the type of control that is being edited. It contains the data-specific attributes for the control. The Control Settings section for the Numeric control is shown in the following figure.

![Control Settings for a Numeric Control](image1)

Figure 2-10. Control Settings for a Numeric Control

Rings and list boxes have a **Label/Value Pairs** button in the Control Settings section. This button activates the **Edit Label/Value Pairs** dialog box shown in the following figure.

![Edit Label/Value Pairs](image2)

Figure 2-11. The Edit Label/Value Pairs Dialog Box
Use the Edit Label/Value Pairs dialog box to create and edit the contents of ring and list box controls. Use the list control functions in the User Interface Library to control rings and list boxes.

The Control Appearance section of the dialog box senses the type of control that is being edited. It contains attributes pertaining to the physical appearance of the control. The Control Appearance section for the Numeric control appears in the following figure.

![Control Appearance](image)

Figure 2-12. Control Appearance for a Numeric Control

The Label Appearance section of the dialog box contains attributes pertaining to the physical appearance of the control label. The Label Appearance section for the Numeric control is shown in the following figure.

![Label Appearance](image)

Figure 2-13. Label Appearance for a Numeric Control

If you type a double underscore before any letter in the label, the user can select the control by pressing <Alt> and the underlined letter, provided that no menu bar is accessible. This feature helps you access controls on pop-up panels.
The Quick Edit Window section of the Edit Control dialog box appears in the following figure.

![Quick Edit Window](image)

Figure 2-14. Quick Edit Window

From the Quick Edit Window, you can perform high level edits on the control. The tools in the tool bar operate like the tools in the main User Interface Editor window. The Quick Edit Window also provides immediate feedback for any changes you make in other section of the dialog box.

Simply stated, the dialog box of any control allows you to interactively set all of the attributes of the control. The Programming with Controls section in Chapter 3, Programming with the User Interface Library, describes these attributes in detail.

**Tab Order**

Each control on a panel has a position in the tab order. The tab order determines which control becomes the next active control when the user presses <Tab> or <Shift-Tab>.
When you create a control, it is added to the end of the tab order. When you copy and paste a control, the new control sits immediately before the source control in the tab order. Select **Tab Order** from the **Edit** menu to put the panel into tab order edit mode as shown in the following figure.

![Edit Tab Order Dialog Box](image)

**Figure 2-15. The Edit Tab Order Dialog Box**

Click on a control with the special pointer cursor, ![special cursor], to change the tab position of a control to the number in the **Click to set to** box. You can change the cursor to the special eyedropper cursor, ![eyedropper cursor], by holding down the `<Ctrl>` key. This eyedropper cursor changes the number in the **Click to set to** box to the current tab position associated with that control. Clicking on ![OK] sets the new tab order. Clicking on ![X] erases the new tab order and keeps the original tab order, which appears in dim display to the right of the new tab order you enter.

**Set Default Font**

The **Set Default Font** command in the **Edit** menu makes the font of the currently selected control the default control font. If the label is also selected (or is the only item selected), the font of the label becomes the default label font. Newly created controls inherit the default fonts.

**Apply Default Font**

The **Apply Default Font** command in the **Edit** menu sets the font of the currently selected control (and/or label) to the default control font (and/or default label font).
Control Style

Use the **Control Style** command to change the style of the selected control. For example, you can change a ring slide control to a ring knob control, and the label/value pairs remain intact.

Create Menu

Commands in the **Create** menu create panels, menu bars, and controls.

Figure 2-16 shows the **Create** menu.

![Create Menu](image)

Figure 2-16. The Create Menu

Panel

The **Panel** command places a new, untitled panel onto the User Interface Editor window. The **Edit Menu** section of this chapter provides details on editing the panel.

Menu Bar…

The **Menu Bar** command brings up the Edit Menu Bar dialog box. The **Edit Menu** section of this chapter provides details on editing the menu.

Controls

The remaining options in the **Create** menu allow you to create GUI controls. You can edit the controls you create from the **Edit** menu. The **Edit Menu** section of this chapter provides details on editing the menu.
View Menu

This section explains how to use the commands in the View menu in the User Interface Editor window.

Figure 2-17 shows the View menu.

![Figure 2-17. The View Menu](image)

Find UIR Objects...

Use the Find UIR Objects command to locate objects in user interface resource file. When you select this command, the Find UIR Objects dialog box opens, as shown in Figure 2-18.

![Figure 2-18. Find UIR Objects Dialog Box](image)

Select the type(s) of objects you want to search for by checking the appropriate check boxes in the left column of the dialog box.

Select the search criterion from the Search By ring control. The choices are:

- Constant Prefix (valid for panels and menu bars)
• Constant Name (valid for controls, menus, and menu items)

• Prefix + Constant Name (valid for all)

• Callback Function Name (valid for all except menu bars)

• Label (valid for all except menu bars)

Enter the text you want to search for into the string control. You can view a list of all the strings in the file which match the current search criterion by clicking on the arrow to the right of the string control, or by using the up and down arrow keys.

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

The **Case Sensitive** option finds only instances of the specified text which match exactly.

If you select **Regular Expression**, LabWindows/CVI treats certain characters in the search string control as regular expression characters instead of literal characters. The regular expression characters are described in Table 4-1 of the *LabWindows/CVI User Manual*.

Press the **Find** button to perform the search. If any user interface objects are matched, the dialog box is replaced by the one shown in Figure 2-19.

![Figure 2-19. Find UIR Objects Dialog Box After a Search Executes](image)

This dialog box allows you to browse through the list of matches. As you come to each object, its callback function name and label are shown, and the object is highlighted.

**Find Prev** searches backward for the previously matched object.

**Find Next** searches for the next matching object.

The **Edit** command terminates the search and brings up the Edit dialog box for the user interface object currently highlighted.

The **Stop** command terminates the search.
Show/Hide Panels

The **Show/Hide Panels** command has a sub-menu, as shown in the following figure.

![Menu](image)

**Figure 2-20. The Show/Hide Panel Sub-Menu**

Use this submenu to select individual panels to view in the User Interface Editor, or to select **Show All Panels** or **Hide All Panels**.

**Bring Panel to Front** command has a submenu that allows you to select a panel from a list to bring to the front for editing.

**Next Panel** brings the next panel in the current .uir file to the front for editing.

**Previous Panel** brings the previous panel in the current .uir file to the front for editing.

**Preview User Interface Header File**

The **Preview User Interface Header File** command brings up a Source code window with a preview of the header file that would be associated with the .uir file in the User Interface Editor window if it were saved.
Arrange Menu

This section explains how to use commands in the Arrange menu in the User Interface Editor window. Figure 2-21 shows the Arrange menu.

![Arrange Menu](image)

**Alignment…**

The Alignment command allows you to align controls on a panel. Use the mouse to select a group of controls by dragging over them or you can <Shift-Click> on each item you want to include in the group. Then you can select an alignment method from the submenu shown in the following figure.

![Alignment Menu](image)

- **Left Edges** vertically aligns the left edges of the selected controls to the left-most control.
- **Horizontal Centers** vertically aligns the selected controls through their horizontal centers.
- **Right Edges** vertically aligns the right edges of the selected controls to the right-most control.
**Top Edges** horizontally aligns the top edges of the selected controls to the upper-most control.

**Vertical Centers** horizontally aligns the selected controls through their vertical centers.

**Bottom Edges** horizontally aligns the bottom edges of the selected controls to the lower-most control.

### Align Horizontal Centers

The **Align Horizontal Centers** command performs the same action as the **Alignment** command, using the alignment option last selected in the **Alignment** command submenu.

### Distribution...

The **Distribution** command allows you to distribute controls on a panel. Select a group of controls by dragging the mouse over them or you can <Shift-Click> on each item you want to include in the group. Then you can select a distribution method from the submenu as shown in the following figure.

![Distribution Submenu](image)

Figure 2-23. The Distribution Submenu

**Top Edges** sets equal vertical spacing between the top edges of the controls. The upper-most and lower-most controls serve as anchor-points.

**Vertical Centers** sets equal vertical spacing between the centers of the controls. The upper-most and lower-most controls serve as anchor-points.

**Bottom Edges** sets equal vertical spacing between the bottom edges of the controls. The upper-most and lower-most controls serve as anchor-points.

**Vertical Gap** sets equal vertical gap spacing between the controls. The upper-most and lower-most controls serve as anchor-points.
**Vertical Compress** compresses the spacing of controls to remove any vertical gap between the controls.

**Left Edges** sets equal horizontal spacing between the left edges of the controls. The left-most and right-most controls serve as anchor-points.

**Horizontal Centers** sets equal horizontal spacing between the centers of the controls. The left-most and right-most controls serve as anchor-points.

**Right Edges** sets equal horizontal spacing between the right edges of the controls. The left-most and right-most controls serve as anchor-points.

**Horizontal Gap** sets equal horizontal gap spacing between the controls. The left-most and right-most controls serve as anchor-points.

**Horizontal Compress** compresses spacing of the controls to remove any horizontal gap between the controls.

**Distribute Vertical Centers**

The **Distribute** command performs the same action as the **Distribution** command according to the option last selected in the **Distribution** command submenu.

**Control ZPlane Order**

The **Control Z-Plane Order** option lets you sets the sequence in which controls are drawn. Controls are always drawn in order, from the back to the front of the z-plane order. The **Control ZPlane Order** submenus presents four commands:

- **Move Forward** moves the control one place forward in the z-plane order.
- **Move Backward** moves the control one place backward in the z-plane order.
- **Move to Front** moves the control to the front of the z-plane order so that it is drawn last.
- **Move to Back** moves the control to the back of the z-plane order so that it is drawn first.

**Center Label**

The Center Label command centers the label of the selected control.
Control Coordinates…

The Control Coordinates command invokes a dialog box allowing you to interactively set the width, height, top, and bottom of all selected controls and labels.

The Code Menu

Use the commands in the Code menu to generate code automatically based on a (.uir) file you are creating or editing. Figure 2-24 shows the Code menu.

![Figure 2-24. The Code Menu](image)

Set Target File…

Use the Set Target File command to specify to which file LabWindows/CVI generates code. Selecting this command brings up the Set Target File dialog box, shown in Figure 2-25. By default, LabWindows/CVI places the generated code in a new window, unless a source code (.c) file is open. Then, that source file is the default target file. CodeBuilder uses the same target file as the function panel target file, except when the function panel target file is the Interactive Execution window.

![Figure 2-25. The Set Target File Dialog Box](image)
To set a target file, select a file from the list of options in the Set Target File dialog box, and then select **OK**. The options include all open source files and a new window.

**Generate**

The commands in the **Generate** menu produce code based on the .uir file. Figure 2-26 shows the **Generate** menu. The code produced by the **Generate** menu uses the bracket styles specified with the **Bracket Styles** command in the Source window’s **Options** menu.

![Figure 2-26. The Generate Menu](image)

**Note:** **Panel Callback** and **Control Callbacks** are enabled when you select any panel or control for which you have specified a callback function name. All **Callbacks** is enabled once you have created any object with a callback function name. **Menu Callbacks** is enabled once you have created a menu bar that contains at least one item with a callback function name.

When you generate code to accompany a .uir file, LabWindows/CVI places the skeleton code in the target file. You must save the .uir file before you can generate any code based on that file. When you save a .uir file, LabWindows/CVI generates a header file (a file with a .h suffix) with the same name. This .h file and userint.h are included in the source file.

If you try to generate the same function more than once, the Generate Code dialog box appears. Figure 2-27 shows the Generate Code dialog box. Each previously generated code fragment appears highlighted. Choose an option in the Generate Code dialog box to replace the existing function, insert a new function, or skip to the next generated function.

![Figure 2-27. The Generate Code Dialog Box](image)
All Code...

Use the **All Code** command to generate code to accompany the .uir file. Selecting **Code » Generate » All Code** brings up the Generate All Code dialog box, shown in Figure 2-28. This dialog box displays a checklist and prompts you to choose the panel(s) the `main` function loads and displays at run time. LabWindows/CVI automatically assigns a default panel variable name for each panel in the .uir file.

![Generate All Code Dialog Box](image)

Figure 2-28. The Generate All Code Dialog Box

The Generate All Code dialog box also prompts you to choose the callback function(s) that terminate the program. For a CodeBuilder program to terminate successfully, you must include a call to the `QuitUserInterface` function.

**Note:** *Callback functions associated with close controls are automatically checked in the Program Termination section of the Generate All Code dialog box. You can specify close controls in the Edit Panel dialog box.*

To automatically generate all code, select the panels you want to load and display in the user interface. Also select the callback function(s) you want to terminate the program, and then select **OK**.
When you choose **Code » Generate » All Code**, LabWindows/CVI produces the `#include` statements, the variable declarations, the function skeletons, and the `main` function, and places them in the target file. The callback function you selected to terminate program execution includes a call to the User Interface Library `QuitUserInterface` function.

Unless you have selected the **Always Append Code to End** option, LabWindows/CVI places the skeleton code for each callback function at the cursor position in the target file. If the cursor is inside an existing function, LabWindows/CVI repositions the cursor at the end of that function before inserting the new function. CodeBuilder places all functions of one type (panel, control, or menu) together in the source file. Any panel callbacks are placed first in the source file, control callbacks are placed next and menu callbacks are placed last. See the *Preferences* section in this chapter for more details on specifying the location of generated code.

Function skeletons for control and panel callbacks include the complete prototype, the proper syntax, a return value and a switch construct containing a case for each default control or panel event. Function skeletons for menu callbacks include the complete prototype and open and close brackets. You can set the default events by selecting **Code » Preferences**. See the *Preferences* section for more details. You can set the location of the open and close brackets by selecting the **Bracket Style** command from the Options menu of a Source window.

**Main Function...**

Use the **Main Function** command to generate code for the `main` function, and write it to the target file. Selecting **Code » Generate » Main Function** option brings up the Generate Main Function dialog box, shown in Figure 2-29. This dialog box prompts you to choose the panel the `main` function loads and displays at run time. LabWindows/CVI automatically assigns a default panel variable name for each panel in the .uir file.

![Figure 2-29. The Generate Main Function Dialog Box](image-url)
Note: If you previously selected Code » Generate » All Code command, you do not need to execute this command as well. Only use this command when you want to replace the main callback function to add or change the panels to be loaded at run time.

To automatically generate code for the main function, select the panel(s) you want to load and display in the user interface, and then select OK.

When you choose Code » Generate » Main Function, LabWindows/CVI produces the #include statements, the variable declarations, and the main callback function, and places them in the target file.

Note: If the source file contains only the main function and the #include statements, and you have not yet created the appropriate callback functions, you might get an error when trying to run the project. When the main function calls LoadPanel, LabWindows/CVI generates a non-fatal error for each callback function it cannot find in the source file.

The checkbox, Generate WinMain() instead of main(), enables you to use WinMain instead of main for your main program. In LabWindows/CVI, you can use either function as your program entry point. When linking your application in an external compiler, it is easier to use WinMain.

If your project target is a DLL, neither WinMain or main are generated. Instead, a DLLMain function is generated. The bulk of the User Interface function calls, however, are generated in a function call InitUIForDLL. You can call InitUIForDLL in your DLL at the point you want to load and display panels.

When you link your executable or DLL in an external compiler, you need to include a call to the InitCVIRTE function in WinMain, main, or DLLMain (or DLLLEntryPoint for Borland C/C++). In a DLL, you also need to include a call to CloseCVIRTE. See the Calling InitCVIRTE and CloseCVIRTE section in Chapter 3, Windows 95 and NT Compiler/Linker Issues, in the LabWindows/CVI Programmer Reference Manual.

CodeBuilder automatically generates the necessary calls to InitCVIRTE and CloseCVIRTE in your WinMain, main, or DLLMain function. It also automatically generates a #include statement for the cvirte.h file.

All Callbacks

Use the All Callbacks command to generate code for all the callback functions, and write them to the target file.

When you select Code » Generate » All Callbacks, LabWindows/CVI produces the #include statements and the callback function skeletons, and places them in the target file.
Panel Callback

Use the **Panel Callback** command to generate code for a panel connected to a callback function. Before you can choose **Code » Generate » Panel Callback**, you must activate a panel.

When you select **Code » Generate » Panel Callback**, LabWindows/CVI produces the `#include` statements and the function skeleton for the active panel, and places them in the target file.

Control Callbacks

Use the **Control Callbacks** command to generate code for a control connected to a callback function. Before you can choose **Generate » Control**, you must select at least one control.

When you select **Code » Generate » Control Callbacks**, LabWindows/CVI produces the `#include` statements and the function skeleton for each selected control, and places them in the target file.

You can also generate a control callback function skeleton by clicking on the control with the right mouse button, and selecting the **Generate Control Callback** command from the pop-up menu.

Menu Callbacks...

Use the **Menu Callbacks** command to generate code for menus and menu items connected to callback functions.

Selecting **Code » Generate » Menu Callbacks** brings up the Select Menu Bar Objects dialog box. Select the menu bar objects for which you want to generate callbacks, and then select **OK**.

When you select **OK**, LabWindows/CVI produces the `#include` statements, the function prototypes and the opening and closing brackets for each callback function. No switch construct or case statements are produced because the usual default events do not apply to Menu Callback functions. You must add the code to implement the actions you want to take place when a menu bar item is selected.

View

Use the **View** command to look at code for a given callback function. Figure 2-30 shows the **View** menu.
To view the code for a function from the .uir file, select a panel or control and then select View » Panel Callback or View » Control Callback. The source file containing the callback function appears with the function name highlighted. You can also view the code for a control callback function by clicking on the control with the right mouse button, and selecting the View Control Callback command from the pop-up menu.

When you choose the View command for a callback function, LabWindows/CVI searches for that function in all open Source windows and in all the source files in the project, and in any other open source files. If the function is found in a closed project file, that file is opened automatically.

The View command is useful because the callback functions for one user interface can be in several different files, and scrolling the source code is not efficient. With the View command, you can move instantly from the user interface file to an object’s callback function whether the source file is open or closed.

When you are finished reviewing the code, you can return instantly to the .uir file from the source file. To return to the .uir file, place the cursor on the callback function name or constant name of the User Interface object you want to go to, and select the Find UI Object command from the View menu in the Source window.

**Note:** You cannot use the View command for Menu Callback functions.

**Preferences**

Use the Preferences command to change the default settings for case statements generated for control callback functions and panel callback functions or to specify the target file location for generated code. Figure 2-31 shows the Preferences submenu.
Default Panel Events… and Default Control Events…

Use the **Default Panel Events** or **Default Control Events** commands to select which events LabWindows/CVI places into the switch construct of the code for panel or control callback functions, respectively. You can choose from several events, and you can choose to add the “default:” switch case. Selecting **Code » Preferences » Default Panel Events** brings up the Panel Callback Events dialog box. Selecting **Code » Preferences » Default Control Events** brings up the Control Callback Events dialog box.

To set the **Default Panel Events** or **Default Control Events**, select the events you want to be included in the code as case statements, and then select **OK**. For each option you choose, LabWindows/CVI includes a case statement that corresponds to this option in the source code.

**Note:** Default control events are ignored for Timer Control Callbacks, whose only event cases are EVENT_TIMER_TICK and EVENT_DISCARD.

Always Append Code to End

When this option is selected, LabWindows/CVI places the skeleton code for each callback function at the end of the target file. When this option is not selected, newly generated code is placed at the current position of the cursor in the target file.

Run Menu

The Run Menu contains a subset of the commands that appear in the Run Menu of the source window. The commands are:

- **Run Project**
- **Continue**
- **Step Over**
- **Step Into**
- **Finish Function**
- **Terminate Execution**
- **Break at First Statement**
- **Breakpoints**

Refer to the *LabWindows/CVI User Manual*, Chapter 4, Source, Interactive Execution, and Standard Input/Output Windows, in the Run Menu section for descriptions of each of these commands.

Library Menu

The **Library** menu for the User Interface Editor Window works the same way as the Library menu in the Project Window. See the *LabWindows/CVI User Manual*, Chapter 3, Project Window, for information on the **Library** menu.
Window Menu

The Window menu in User Interface Editor windows behaves like the Window menu in the Project window. See the LabWindows/CVI User Manual, Chapter 3, Project Window, for information on the Window menu.

Options Menu

This section explains how to use the commands in the Options menu. Figure 2-32 shows the Options menu.

![Options Menu](image)

Figure 2-32. The Options Menu

Operate Visible Panels

Operate Visible Panels allows you to operate the visible panels as you would in an application program. This command has the same effect as clicking on the icon. When you finish operating the panel, select Operate Visible Panels again to return to edit mode.

Next Tool

The Next Tool command in the Options menu cycles the User Interface Editor through three of its four modes: editing mode, labeling mode, and coloring mode.
Preferences…

Selecting **Preferences** brings up the User Interface Editor Preferences dialog box, as shown in the following figure.

![User Interface Editor Preferences](image)

**Figure 2-33. The User Interface Preferences Dialog Box**

**Undo Preferences**

Use the Undo Preferences section of the User Interface Editor Preferences dialog box to set the number of undo-able actions for each file. Select the **Purge undo actions when saving file** check box if you want the Undo buffer to empty every time you save a file.

**Color Preferences**

Use the Color Preferences section of the User Interface Editor Preferences dialog box to set the default editor and panel colors.

**Constant Name Assignment**

Constant names link user GUI objects and your program. The User Interface Editor writes all assigned constant names to a header file corresponding to the loaded .uir file.
The Constant Name Assignment section of the User Interface Editor Preferences dialog box allows you to set preferences for constant name assignment, when you do not assign constant names yourself.

When the "Immediately assign constant names for new objects" option is in effect, the User Interface Editor generates constant names for each object as you create it. For panels and controls, the generated constant name appears in the edit dialog the first time you bring it up. For menu bars, the constant names are assigned only when you exit the menu bar editor. In all cases, you can freely modify the generated constant names.

It is recommended that you leave the "Immediately assign constant names for new objects" option in effect. This makes it easier for you to use the other LabWindows/CVI features that have been designed to help you write your program to operate your user interface.

Notice that when the "Immediately assign constant names for new objects" option is enabled, the "Assign constant names from user-defined control labels when possible" option has no effect. That is because the control labels have not been changed from the default "Untitled Control" label at the time the constant name is generated. Instead, the constant name is based on the control type.

If you choose to disable the "Immediately assign constant names for new objects" option, it is recommended that you enable the "Fill in missing constant names when saving" option.

**Assign Missing Constants…**

The Assign Missing Constants command assigns constant names to all of the objects in the User Interface Editor Window that currently do not have constant names. A confirmation dialog appears showing the number of items that have no constant names.

**Save In Text Format**

The Save In Text Format command saves the contents of the User Interface Editor Window in a ASCII text format. A dialog box appears prompting you to enter the pathname under which to save the text file. The extension .tui is recommended for such files. Do NOT use the .uir extension.

The ASCII text file contains descriptions of all the objects in the User Interface Editor Window.

Note: *If you have a large number of objects in your User Interface Editor Window, this command may take a significant amount of time to complete (for example, 3 minutes).*
Load From Text Format…

The Load From Text Format command loads into a new User Interface Editor Window the objects defined in a file saved using the Save In Text Format command. A dialog box appears prompting you for the pathname of the file.
Chapter 3
Programming with the User Interface Library

This chapter describes how to use the User Interface Library in the application programs you create.

Developing and Running a Program

Although there are many ways to develop your projects in LabWindows/CVI, you may want to use the following development pattern.

1. Open a User Interface Editor window to design a user interface for your program.

2. Assign constant names and/or callback functions to each control on your GUI.

3. Save your GUI as a user interface (.uir) file (the program automatically generates a corresponding .h include file). If you use CodeBuilder, LabWindows/CVI creates skeleton code for your source file. See the CodeBuilder Overview section of Chapter 2, User Interface Editor Reference, for more information.

4. Open a Source window to develop your C Language program to control the GUI.

5. Add the source code (.c), include (.h), and user interface (.uir) files to your project list and save as a project (.prj) file.

If you are editing the program and .uir file concurrently, you must recompile your program every time you save the .uir file. This is required because the contents of the include file generated by the User Interface Editor might change. You can make recompiling automatic by enabling the Track Include File Dependencies command in the Options menu of the Project Window. If the .uir files and image files do not contain absolute paths in the LoadMenuBar, LoadPanel, or DisplayImageFile functions, the User Interface Library will search for them in the following order.

1. The directory associated with the .uir or image file in the project.

2. The directory containing the project.

It is good practice to add your .uir and image files to the project list or save the resource files and image files in the directory that contains your project file. If you use the latter approach, your program should not contain absolute path names in the LoadMenuBar, LoadPanel or DisplayImageFile functions.
Creating a Graphical User Interface

As discussed in the *Introduction to the Graphical User Interface* section of Chapter 1, *User Interface Concepts*, there are two ways to create Graphical User Interface (GUI) objects for your application program. You can create objects programmatically using function calls or interactively using the User Interface Editor.

Once you create a GUI, you can control the objects in two ways. You can assign callback functions to GUI objects. When any type of event is generated on a panel, menu, or control, the appropriate callback function executes. Alternatively, you can use an event loop that includes a call to *GetUserEvent*. When a commit event is generated, *GetUserEvent* returns the appropriate panel, menu, or control identifier, and the program conditionally executes portions of code. You can use either technique or you can combine them for added flexibility.

Naturally, one of your most important tasks in designing a GUI is to assign callback functions and IDs to every interface object.

**If you design your GUI in the User Interface Editor...**

You assign callback functions and unique IDs (constant names) to objects and LabWindows/CVI automatically saves their declarations in an include file whenever you save the resource file. You must include this file in your application program using the `#include` preprocessor command. The `#include` directive allows the program to reference the resource IDs and callback functions for the user interface objects. Panel and menu bar handles are assigned at runtime when you use the *LoadPanel* and *LoadMenuBar* functions.

Note:  *With the LabWindows/CVI Code Builder, you can create automatically complete C code that compiles and runs based on a user interface (.uir) file you are creating or editing. See the Code Builder Overview section of Chapter 2, User Interface Editor Reference, for more information.*

**If you design your GUI programmatically...**

Assign panel and menu bar handles with the *NewPanel*, and *NewMenuBar* functions. Assign IDs using the *NewCtrl*, *NewMenu*, *NewSubMenu*, and *NewMenuItem* functions. Assign callback functions through functions such as *InstallPanelCallback*, *InstallCtrlCallback*, and *InstallMenuCallback*.

**Assigning Constant Names in the User Interface Editor**

The following rules apply when you use the User Interface Editor to assign constant names to panels and controls.

- You must assign a constant prefix to the panel. Panel constant prefixes can be up to 10 characters long and must be unique with respect to all other panel and menu bar constant
prefixes in the same resource file. Your application program references the prefix when it loads the panel from the resource file.

- You may assign a constant name to each control. Control constant names can be up to 20 characters long and must be unique with respect to all other control constants defined for the same panel. The name is concatenated to the panel prefix to generate a unique constant name. For example, if the panel prefix is `SCOPE` and the control prefix is `POWER`, the complete constant name would be `SCOPE_POWER`. The application program uses the complete constant name to reference the control.

The following rules apply when assigning constant names to menu bars in the User Interface Editor.

- You must assign a constant prefix to the menu bar. The prefix can be up to 10 characters long and must be unique with respect to all other panel and menu bar prefixes that you store together in a resource file. Your application program references the prefix to load the menu bar from the resource file.

- You may assign a constant prefix to each menu. The prefix can be up to 10 characters long and must be unique with respect to all other menu prefixes in the same menu bar. The program concatenates the constant prefix and the menu bar prefix to generate a unique constant name. For example, if the menu bar prefix is `MAIN` and the menu prefix is `FILE`, the complete constant name is `MAIN_FILE`. Your application program uses the complete constant name to reference the menu.

- You may assign a constant name to each menu item. Menu item constant names can be up to 10 characters long and must be unique with respect to all other menu item constants in the same menu. The program concatenates the menu bar prefix and the menu prefix to generate a unique constant name. For example, if the menu bar prefix is `MAIN`, the menu prefix is `FILE`, and the menu item name is `LOAD`, the complete constant name is `MAIN_FILE_LOAD`. Your application program uses the complete constant name to reference the menu item.

The User Interface Editor automatically inserts the constant name separator (`_`) when it generates the include file.

Note: When you use submenus, all identifiers starting with the menu bar prefix are concatenated. Choose brief identifiers so that constant names remain as short as possible.

**Controlling a Graphical User Interface**

Certain user operations on the GUI, such as selecting a menu item on the GUI or typing a value, are called *events*. The User Interface Library provides the link between events and the code files in your project.
User Interface Events

Your program can recognize events and execute the code in response to them. Table 3-1 shows all of the events that are generated from the GUI and the information that is passed to the program at event time.

Table 3-1. User Interface Events

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event on the GUI</th>
<th>Information Passed to Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>control and menu event</td>
<td>EVENT_COMMIT</td>
<td>Which panel or menu bar, which control or menu item.</td>
</tr>
<tr>
<td>control event</td>
<td>EVENT_VAL_CHANGED</td>
<td>Which panel, which control.</td>
</tr>
<tr>
<td>control and panel event</td>
<td>EVENT_LEFT_CLICK</td>
<td>Which panel, which control, mouse y &amp; x coordinates.</td>
</tr>
<tr>
<td></td>
<td>EVENT_LEFT_DOUBLE_CLICK</td>
<td>Which panel, which control, mouse y &amp; x coordinates.</td>
</tr>
<tr>
<td></td>
<td>EVENT_RIGHT_CLICK</td>
<td>Which panel, which control, mouse y &amp; x coordinates.</td>
</tr>
<tr>
<td></td>
<td>EVENT_RIGHT_DOUBLE_CLICK</td>
<td>Which panel, which control, mouse y &amp; x coordinates.</td>
</tr>
<tr>
<td></td>
<td>EVENT_KEYPRESS</td>
<td>Which panel, which control, key code, pointer to key code.</td>
</tr>
<tr>
<td></td>
<td>EVENT_GOT_FOCUS</td>
<td>Which panel, which control</td>
</tr>
<tr>
<td></td>
<td>EVENT_LOST_FOCUS</td>
<td>Which panel, which control</td>
</tr>
<tr>
<td></td>
<td>EVENT_DISCARD</td>
<td>Which panel, which control</td>
</tr>
<tr>
<td>timer control event</td>
<td>EVENT_TIMER_TICK</td>
<td>Pointer to the current time (double *), pointer to time since the callback last received an EVENT_TIMER_TICK(double *)</td>
</tr>
<tr>
<td>panel event</td>
<td>EVENT_CLOSE</td>
<td>Which panel.</td>
</tr>
<tr>
<td></td>
<td>EVENT_PANEL_SIZE</td>
<td>Which panel.</td>
</tr>
<tr>
<td></td>
<td>EVENT_PANEL_MOVE</td>
<td>Which panel.</td>
</tr>
<tr>
<td>main callback event</td>
<td>EVENT_IDLE</td>
<td>Obsolete. Use timer controls instead.</td>
</tr>
<tr>
<td></td>
<td>EVENT_END_TASK</td>
<td>Microsoft Windows only. Received when Windows wants to quit. (Return a non-zero value to abort the termination.)</td>
</tr>
</tbody>
</table>

© National Instruments Corporation
Using Callback Functions to Respond to User Interface Events

Callback functions respond to all events listed in Table 3-1. Callback functions have prototypes (found in `userint.h`) for panels, menu bars, controls, or the main callback. When the user generates an event on a particular user interface object, the appropriate callback function executes. Idle events and end-task events are passed to the main callback function only. (See the discussion of `InstallMainCallback` in the *Special User Interface Functions* section of Chapter 3, *Programming With the User Interface Library*. Event information is passed automatically from the GUI to your program through callback functions whenever they are called. For example, callback functions are passed the user interface event type that occurred (such as `EVENT_LEFTCLICK`), and some additional information concerning that event (such as the x and y-coordinates of the mouse cursor when the click occurred). You, as the developer of these callback functions, are free to use this information as needed when responding to events.

Callback functions are introduced in the tutorial in the *Getting Started with LabWindows/CVI* manual. Many of the sample programs described in Chapter 5, *LabWindows/CVI Sample Programs*, illustrate callback functions, too. The following diagram and example pseudo-code illustrates the callback function concept.

![The Callback Function Concept](image)

Figure 3-1. The Callback Function Concept
panel_handle = LoadPanel(...)  
DisplayPanel(panel_handle, ...);  
menu_handle = LoadMenuBar(...);  
RunUserInterface()  

int CVICALLBACK PanelResponse (int handle, int event, void *callbackdata,  
     int eventdata1, int eventdata2)  
{  
    switch (event) {  
      case EVENT_PANEL_SIZE :  
        /* Code that responds to the panel */  
        /* being resized */  
        break;  
      case EVENT_PANEL_MOVE :  
        /* Code that responds to the panel */  
        /* being moved */  
        break;  
      case EVENT_KEYPRESS :  
        /* Code that responds to a keypress */  
        /* eventdata1 & eventdata2 contain */  
        /* keycode information */  
        break;  
    }  
    return(0);  
}  

int CVICALLBACK ControlResponse (int handle, int control, int event,  
     void *callbackdata, int eventdata1, int eventdata2)  
{  
    if (control == PANEL_CONTROL1) {  
      switch (event) {  
        case EVENT_RIGHT_CLICK :  
          /* Code that responds to a right */  
          /* click on PANEL_CONTROL1 */  
          break;  
        case EVENT_VAL_CHANGED :  
          /* Code that responds to a value */  
          /* change on PANEL_CONTROL1 */  
          break;  
        case EVENT_COMMIT :  
          /* Code that responds to a commit */  
          /* event on PANEL_CONTROL1 */  
          break;  
      }  
    }  
    if (control == PANEL_CONTROL2) {  
      switch (event) {  
        case EVENT_RIGHT_CLICK :  
          /* Code that responds to a right */  
          /* click on PANEL_CONTROL2 */  
          break;  
        case EVENT_COMMIT :  
          /* Code that responds to a commit */  
          /* event on PANEL_CONTROL2 */  
          break;  
      }  
    }  
    return(0);  
}
int CVICALLBACK MenuBarResponse (int menubar, int menuitem,
   void *callbackdata, int panel)
{
   switch (menuitem) {
      case MENUBAR_MENU1_ITEM1:
         /* Code that responds to ITEM1 in MENUBAR_MENU1 of the menu bar. */
         break;
      case MENUBAR_MENU1_ITEM2:
         /* Code that responds to ITEM2 in MENUBAR_MENU1 of the menu bar. */
         break;
   }
   return(0);
}

Note: If you assign callback functions to your GUI objects using the User Interface Editor, these functions are automatically installed through LoadPanel and LoadMenuBar. Otherwise, you will need to install them programmatically through the Install functions (InstallPanelCallback, InstallCtrlCallback, InstallMenuCallback, and others).

Note: Do not call the longjmp function from within a callback function.

The CVICALLBACK macro should precede the function name in the declarations and function headers for all user interface callbacks. This ensures that the functions are treated by the compiler as cdecl (or stack-based in WATCOM), even when the default calling convention is stdcall. CVICALLBACK is defined in cvidefs.h, which is included by userint.h. The CVICALLBACK macro is included where necessary in the header files generated by the User Interface Editor and in source code generated by CodeBuilder.

For detailed information about the user interface callback functions, see the descriptions of PanelCallbackPtr, CtrlCallbackPtr, MenuCallbackPtr, MenuDimmerCallbackPtr, MainCallbackPtr, and DeferredCallbackPtr in Chapter 4, User Interface Library Reference.
Using GetUserEvent to Respond to User Interface Events

The GetUserEvent function returns only commit events (with the exception of programmer-defined events posted through QueueUserEvent). Commit events are generated when the user of the GUI actually commits to an operation such as making a menu selection or typing in a menu selection or typing in a number and pressing <Enter>. The following figure illustrates the event-loop concept of GetUserEvent.

A typical program could use a GetUserEvent loop with the following pseudocode algorithm.

```c
panel_handle = LoadPanel(...);
DisplayPanel(panel_handle,...);
menu_handle = LoadMenuBar(...);

while (GetUserEvent(WAIT, &handle, &control)) {
    if (handle == PANEL) {
        switch (control) {
            case PANEL_CONTROL1:
                /* Code that responds to CONTROL1 on the panel. */
                break;
        }
    }
}
```
case PANEL_CONTROL2:
  .  /* Code that responds to CONTROL2 on */
  .  /* the panel.                          */
  break;
}

if (handle == MENUBAR)  {
  switch (control)  {
    case MENUBAR_MENU1_ITEM1:
      .  /* Code that responds to ITEM1 in */
      .  /* MENU1 of the menu bar.          */
      break;
    case MENUBAR_MENU1_ITEM2:
      .  /* Code that responds to ITEM2 in */
      .  /* MENU1 of the menu bar.          */
      break;
  }
}

You should read the remainder of this chapter and Chapter 3, *Programming with the User Interface Library*, before attempting to develop a program with the User Interface Library. You should also examine and execute the example programs outlined in Chapter 5, *LabWindows/CVI Sample Programs*. These examples are designed to illustrate the concepts presented in Chapters 1 and 3.

### Programming with Panels

This section describes how you can use the User Interface Library functions to control the elements of user interface panels.

#### Panel Functions

Use `LoadPanel` to load a panel from a resource file created in the User Interface Editor into memory. When `LoadPanel` loads a panel from a resource file, it references the panel using the constant name that you assigned to the panel in the User Interface Editor. `LoadPanel` returns a handle that you use in subsequent User Interface Library functions to reference the panel. Use the first parameter of `LoadPanel` to specify whether the panel loads as a top-level window or as a child of another (parent) window. Loading a panel does not automatically display the panel.

Use `NewPanel` to create a new panel during program execution. `NewPanel` returns a handle that you use in subsequent User Interface Library functions to reference the panel. Use the first parameter of `NewPanel` to specify whether the panel is created as a top-level window or as a child of another (parent) window. You also specify the name, position, and size of the panel through parameters to `NewPanel`. Creating a new panel using `NewPanel` does not automatically display the panel.

Use `DuplicatePanel` to create a new panel that is a duplicate of a panel loaded by `LoadPanel` or created by `NewPanel`. `DuplicatePanel` returns a handle you use to reference the panel in subsequent operations. Use the first parameter of `DuplicatePanel` to specify whether the panel is created as a top-level window or as a child of another (parent) window.
window. You also specify the name and position of the panel through parameters to 
DuplicatePanel. Creating a new panel using DuplicatePanel does not automatically 
display the panel.

Use DisplayPanel to display a panel. When a panel is visible and enabled, the user can 
operate it. Calling DisplayPanel when a panel is already displayed causes the panel to be 
completely redrawn.

Use HidePanel to hide a panel. When a panel is hidden, it can still be updated. For example, 
you can add plots to a graph control on a hidden panel. When the panel is redisplayed, it will 
show the new plots. A hidden panel cannot generate events.

Use DefaultPanel to restore all panel controls to their default values. If the panel is visible, 
it is updated to reflect the new control values. You assign default values to controls in the User 
Interface Editor or through the SetCtrlAttribute function using ATTR_DFLT_VALUE.

Use GetActivePanel to obtain the handle of the currently active panel.

Use SetActivePanel to make a particular panel active when multiple panels are visible.

Use GetPanelAttribute to obtain a particular attribute of a panel. The Panel Attributes 
section of this chapter lists panel attributes.

Use SetPanelAttribute to set a particular attribute of a panel. The Panel Attributes 
section of this chapter lists panel attributes.

Use SetPanelPos to change the position of a panel on the screen. If the panel is visible, it is 
redrawn in its new position.

Use SavePanelState to save the state of a panel to a file on disk. SavePanelState 
saves the state of every control, including the plot data in graphs and strip charts and the current 
items in selection lists.

Use RecallPanelState to recall a state file from disk to a panel. Every control on the panel 
is set to the value stored in the state file. You must recall the state file to the same panel from 
which it was saved.

Use DiscardPanel to remove a panel from memory. If the panel is visible, it is removed 
from the screen.
Role of Child Panels

You can configure the `LoadPanel` function so that panels appear within other panels in your user interface. When you do this, the principal panel is called the parent panel; panels within are child panels. Child panels can appear within other child panels, too.

A child panel helps developers control the appearance of the user interface. Users cannot drag a child panel outside its parent panel. And if users shrink a parent panel, a child panel may be partially or completely hidden in the shrunken panel view.

Processing Panel Events

If you need to process events such as mouse clicks, panel moves, or panel sizing, you must assign a callback function to the panel. When an event is generated on the panel, the panel callback function executes. To process events generated on the controls of the panel, see the Programming with Controls section of this chapter.

If you create your panel in the User Interface Editor, you can assign a callback function name to the panel from within the editor. When you load the panel with `LoadPanel`, LabWindows/CVI automatically installs your callback function and calls it whenever an event is generated on the panel.

If you create your panel programmatically using the `NewPanel` function, you can install a callback function for the panel using `InstallPanelCallback`. Your callback function is called whenever an event is generated on the panel. You should use the `PanelCallbackPtr` typed from `userint.h` as a model to declare your panel callback function.

Callbacks must be initiated through a call to `RunUserInterface` or through a `GetUserEvent` loop.

Panel Attributes

Table 3-2 lists panel attributes that you can retrieve or change through `GetPanelAttribute` and `SetPanelAttribute`. 
### Table 3-2. Panel Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ACTIVE</td>
<td>int</td>
<td>Indicates whether the panel or one of its child panels is the active panel. (GetPanelAttribute only.)</td>
</tr>
</tbody>
</table>
| ATTR_ACTIVATE_WHEN_CLICKED_ON            | int       | 1 = child panel is made the active panel when clicked on  
0 = child panel is not made the active panel when clicked on |
| ATTR_BACKCOLOR                           | int       | RGB value—See discussion following this table.                                                                                              |
| ATTR_CALLBACK_DATA                       | void *    | A pointer to user data that is passed to the panel callback function                                                                         |
| ATTR_CALLBACK_FUNCTION_POINTER           | void *    | A pointer to the callback function for the panel.                                                                                              |
| ATTR_CALLBACK_NAME                       | char *    | The name of the callback function associated with the panel. (GetPanelAttribute only.)                                                       |
| ATTR_CALLBACK_NAME_LENGTH                | int       | The number of characters in the name of the panel callback. (GetPanelAttribute only.)                                                      |
| ATTR_CAN_MAXIMIZE                        | int       | 1 = you can maximize the top-level panel  
0 = you cannot maximize the top-level panel (Windows only)                                                                                   |
| ATTR_CAN_MINIMIZE                        | int       | 1 = you can minimize the top-level panel  
0 = you cannot minimize the top-level panel (Windows only)                                                                                   |
| ATTR_CLOSE_CTRL                          | int       | control ID that receives commit events when a user selects the Close command in the System Menu (Top-level panels only.)                   |
| ATTR_CLOSE_ITEM_VISIBLE                  | int       | Specify whether the panel Close item is available in the top-level panel.  
0 = Dimmed Close Item  
1 = Enable Close Item (Windows only)                                                                                                          |

(continues)
Table 3-2. Panel Attributes (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CONFORM_TO_SYSTEM</td>
<td>int</td>
<td>Specifies whether the panel and its controls use the system colors. Subsequent new controls also use the system colors. (This is useful only on Windows 95. Other platforms always use panel gray and black as the system colors.)</td>
</tr>
<tr>
<td>ATTR_CONSTANT_NAME</td>
<td>char *</td>
<td>The constant name assigned to the panel in the User Interface Editor. (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_CONSTANT_NAME_LENGTH</td>
<td>int</td>
<td>The number of characters in the constant name of the panel. (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_DIMMED</td>
<td>int</td>
<td>1 = panel disabled 0 = panel enabled</td>
</tr>
<tr>
<td>ATTR_FIRST_CHILD</td>
<td>int</td>
<td>The first child panel for the parent panel. (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_FLOATING</td>
<td>int</td>
<td>Specifies whether the panel floats above all non-floating panels. Applies to top-level panels in Microsoft Windows.</td>
</tr>
<tr>
<td>ATTR_FRAME_COLOR</td>
<td>int</td>
<td>RGB value—See discussion following this table. (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_FRAME_STYLE</td>
<td>int</td>
<td>The frame style of the panel. (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_FRAME_THICKNESS</td>
<td>int</td>
<td>Thickness of panel frame. range = 1 to 10 (see Figure 3-3) (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_HEIGHT</td>
<td>int</td>
<td>Height of panel, range = 1 to 32767 (see Figure 3-3)</td>
</tr>
<tr>
<td>ATTR_HSCROLL_OFFSET</td>
<td>int</td>
<td>The offset of the horizontal scroll bar (in pixels)</td>
</tr>
<tr>
<td>ATTR_HSCROLL_OFFSET_MAX</td>
<td>int</td>
<td>The number of pixels you have to scroll to reach the right edge of the right-most object on the panel (GetPanelAttribute only)</td>
</tr>
</tbody>
</table>

(continues)
### Table 3-2. Panel Attributes (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_LEFT</td>
<td>int</td>
<td>Left position of panel, range = 1 to 32767 or VAL_AUTO_CENTER (see Figure 3-3)</td>
</tr>
<tr>
<td>ATTR_MENU_BAR_VISIBLE</td>
<td>int</td>
<td>1 = menu bar visible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = menu bar invisible</td>
</tr>
<tr>
<td>ATTR_MENU_HEIGHT</td>
<td>int</td>
<td>Height of menu bar, range = 1 to 32767 (See Figure 3-3.) (GetPanelAttribute only)</td>
</tr>
<tr>
<td>ATTR_MOUSE_CURSOR</td>
<td>int</td>
<td>The mouse cursor style.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(See Table 3-4.)</td>
</tr>
<tr>
<td>ATTR_MOVABLE</td>
<td>int</td>
<td>1 = movable panel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = immovable panel (child panels only)</td>
</tr>
<tr>
<td>ATTR_NEXT_PANEL</td>
<td>int</td>
<td>The next child panel for the parent panel (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_NUM_CHILDREN</td>
<td>int</td>
<td>The number of child panels for the parent panel (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_NUM_CTRLS</td>
<td>int</td>
<td>The number of controls on the panel (GetPanelAttribute only).</td>
</tr>
<tr>
<td>ATTR_PANEL_FIRST_CTRL</td>
<td>int</td>
<td>The first control on the panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PANEL_MENU_BAR_CONSTANT</td>
<td>char *</td>
<td>The menubar control ID name for the panel. (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PANEL_MENU_BAR_CONSTANT_LENGTH</td>
<td>int</td>
<td>The number of characters in the menubar control ID name. (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PANEL_PARENT</td>
<td>int</td>
<td>The panel constant ID of the parent of the child panel. (GetPanelAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PARENT_SHARES_SHORTCUT_KEYS</td>
<td>int</td>
<td>1 = share shortcut keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = do not share shortcut keys</td>
</tr>
<tr>
<td>ATTR_SCROLL_BAR_COLOR</td>
<td>int</td>
<td>RGB value—See discussion following this table.</td>
</tr>
</tbody>
</table>

(continues)
### Table 3-2. Panel Attributes (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_SCROLL_BARS</td>
<td>int</td>
<td>VAL_NO_SCROLL_BARS or VAL_HORIZ_SCROLL_BAR or VAL_VERT_SCROLL_BAR or VAL_BOTH_SCROLL_BARS</td>
</tr>
<tr>
<td>ATTR_SIZABLE</td>
<td>int</td>
<td>1 = panel sizable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = panel not sizable</td>
</tr>
<tr>
<td>(Child panels only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTR_SYSTEM_MENU_VISIBLE</td>
<td>int</td>
<td>1 = The system menu is visible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = The system menu is not visible</td>
</tr>
<tr>
<td>ATTR_SYSTEM_WINDOW_HANDLE</td>
<td>int</td>
<td>Returns a number that can be cast to obtain the system specific window handle for a top-level window. (GetPanelAttribute only—see following discussion)</td>
</tr>
<tr>
<td>ATTR_TITLE</td>
<td>char *</td>
<td>The title of the panel</td>
</tr>
<tr>
<td>ATTR_TITLE_BACKCOLOR</td>
<td>int</td>
<td>RGB value—See discussion following this table. (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_TITLE_BOLD</td>
<td>int</td>
<td>1 = title is bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = title is not bold</td>
</tr>
<tr>
<td>(Child panels only.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTR_TITLE_COLOR</td>
<td>int</td>
<td>RGB value—See discussion following this table. (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_TITLE_FONT</td>
<td>char *</td>
<td>The font of the panel title (see Table 3-5) (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_TITLE_FONT_NAME_LENGTH</td>
<td>int</td>
<td>Number of characters in the name of the title font (GetPanelAttribute only) (Child panels only.)</td>
</tr>
<tr>
<td>ATTR_TITLE_ITALIC</td>
<td>int</td>
<td>1 = title in italics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = title not in italics</td>
</tr>
<tr>
<td>(Child panel only.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTR_TITLE_LENGTH</td>
<td>int</td>
<td>The number of characters in panel title (GetPanelAttribute only).</td>
</tr>
<tr>
<td>ATTR_TITLE_POINT_SIZE</td>
<td>int</td>
<td>Point size of the title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>range = 1 to 32767</td>
</tr>
</tbody>
</table>

(continues)
Table 3-2. Panel Attributes (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ATTR_TITLE_SIZE_TO_FONT       | int    | Specify whether the child panel title is sized to the font.  
                                  |        | 0 = not sized  
                                  |        | 1 = sized  
| ATTR_TITLE_STRIKEOUT         | int    | 1 = title has strikeout  
                                  |        | 0 = title does not have strikeout  
                                  |        | (Child panels only.)  
| ATTR_TITLE_UNDERLINE         | int    | 1 = title underlined  
                                  |        | 0 = title not underlined  
                                  |        | (Child panels only.)  
| ATTR_TITLEBAR_THICKNESS      | int    | Thickness of panel title bar, range = 1 to 32767 (see Figure 3-3)  
                                  |        | (Child panels only.)  
| ATTR_TITLEBAR_VISIBLE       | int    | 1 = title bar visible  
                                  |        | 0 = title bar invisible  
| ATTR_TOP                     | int    | Top position of panel, range = 1 to 32767 or VAL_AUTO_CENTER. (See Figure 3-3.)  
| ATTR_VISIBLE                 | int    | 1 = panel visible  
                                  |        | 0 = panel invisible  
| ATTR_VSCROLL_OFFSET         | int    | The offset of the vertical scroll bar (in pixels)  
| ATTR_VSCROLL_OFFSET_MAX     | int    | The number of pixels you have to scroll to reach the bottom edge of the lowest object on the panel. (GetPanelAttribute only)  
| ATTR_WIDTH                   | int    | The width of the panel in pixels. Valid range: 0 to 32767.  
| ATTR_WINDOW_ZOOM             | int    | VAL_MAXIMIZE, VAL_MINIMIZE, or VAL_NO_ZOOM  
                                  |        | (Windows only—See following discussion.)  
| ATTR_ZPLANE_POSITION        | int    | The drawing order of the panel. The lowest ordered control (0) is on top. Valid Range: 0 to (Number of Panels – 1) (For child panels only.) |
Panel Attribute Discussion

ATTR_CAN_MAXIMIZE, ATTR_CAN_MAXIMIZE, and ATTR_CLOSE_ITEM_VISIBLE are only implemented for top level windows on Windows. Setting these attributes on child panels or on top level XWindows panels has no effect and does not return errors.

ATTR_SYSTEM_WINDOW_HANDLE returns an integer that can be cast to obtain the system specific window handle for a panel. The attribute returns 0 for child panels. The actual type of the value is HWND on Windows, and Window on XWindows. Setting another panel attribute (example: ATTR_TITLEBAR_VISIBLE) may cause the value of ATTR_SYSTEM_WINDOW_HANDLE to change since the panel has to destroy and recreate its window to change certain attributes. Installing and uninstalling a panel as a pop-up can also change the window handle. So, obtain the window handle again after taking any of these actions.

ATTR_WINDOW_ZOOM is only implemented for top level windows on Windows. Setting it on a child panel or on a top level XWindows panel has no effect and does not return an error.

Setting the ATTR_WINDOW_ZOOM attribute automatically makes the panel visible. If the attribute is set to VAL_NO_ZOOM or VAL_MAXIMIZE, the panel is activated. If the attribute is set to VAL_MINIMIZE, the panel is deactivated. Making the panel invisible automatically resets the ATTR_WINDOW_ZOOM attribute to VAL_NO_ZOOM.

An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. Common colors appear in the following table. The first sixteen colors listed are the sixteen standard colors.

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_RED</td>
<td>0xFF0000L</td>
</tr>
<tr>
<td>VAL_GREEN</td>
<td>0x00FF00L</td>
</tr>
<tr>
<td>VAL_BLUE</td>
<td>0x0000FFL</td>
</tr>
<tr>
<td>VAL_CYAN</td>
<td>0x00FFFFL</td>
</tr>
<tr>
<td>VAL_MAGENTA</td>
<td>0xFFFF00L</td>
</tr>
<tr>
<td>VAL_DK_RED</td>
<td>0x800000L</td>
</tr>
<tr>
<td>VAL_DK_BLUE</td>
<td>0x000080L</td>
</tr>
<tr>
<td>VAL_DK_GREEN</td>
<td>0x008000L</td>
</tr>
<tr>
<td>VAL_DK_CYAN</td>
<td>0x008080L</td>
</tr>
<tr>
<td>VAL_DK_MAGENTA</td>
<td>0x800080L</td>
</tr>
</tbody>
</table>

Table 3-3. Common Color Values (continues)
Table 3-3. Common Color Values (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_DK_YELLOW</td>
<td>0x808000L</td>
</tr>
<tr>
<td>VAL_LT_GRAY</td>
<td>0xCCCCCCL</td>
</tr>
<tr>
<td>VAL_DK_GRAY</td>
<td>0x808080L</td>
</tr>
<tr>
<td>VAL_BLACK</td>
<td>0x000000L</td>
</tr>
<tr>
<td>VAL_WHITE</td>
<td>0xFFFFFFL</td>
</tr>
<tr>
<td>VAL_PANEL_GRAY</td>
<td>0xCCCCCCL</td>
</tr>
<tr>
<td>VAL_GRAY</td>
<td>0xA0A0A0L</td>
</tr>
<tr>
<td>VAL_OFFWHITE</td>
<td>0xE5E5E5L</td>
</tr>
<tr>
<td>VAL_TRANSPARENT</td>
<td>0x1000000L</td>
</tr>
</tbody>
</table>

You can also use the User Interface Library function, MakeColor, to create an RGB value from red, green, and blue color components.

The following list presents ATTR_FRAME_STYLE values.

VAL_OUTLINED_FRAME
VAL_BEVELLED_FRAME
VAL_RAISED_FRAME
VAL_HIDDEN_FRAME
VAL_STEP_FRAME
VAL_RAISED_OUTLINE_FRAME

To see the different panel frame styles, edit a panel in the User Interface Editor and select the various frame styles.

Figure 3-3 shows the geometric attributes of a panel.

Figure 3-3. The Geometric Attributes of a Panel

Note: Always set ATTR_FRAME_STYLE before setting ATTR_FRAME_THICKNESS.
Table 3-4 lists the values and cursor styles corresponding to `ATTR_MOUSE_CURSOR`.

Table 3-4. Values and Cursor Styles for `ATTR_MOUSE_CURSOR`

<table>
<thead>
<tr>
<th>Value</th>
<th>Cursor Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_DEFAULT_CURSOR</td>
<td><img src="image1" alt="default cursor" /></td>
</tr>
<tr>
<td>VAL_CHECK_CURSOR</td>
<td><img src="image2" alt="check cursor" /></td>
</tr>
<tr>
<td>VAL_CROSS_HAIR_CURSOR</td>
<td><img src="image3" alt="cross-hair cursor" /></td>
</tr>
<tr>
<td>VAL_BOX_CURSOR</td>
<td><img src="image4" alt="box cursor" /></td>
</tr>
<tr>
<td>VAL_POINTING_FINGER_CURSOR</td>
<td><img src="image5" alt="pointing finger cursor" /></td>
</tr>
<tr>
<td>VAL_OPEN_HAND_CURSOR</td>
<td><img src="image6" alt="open hand cursor" /></td>
</tr>
<tr>
<td>VAL_QUESTION_MARK_CURSOR</td>
<td><img src="image7" alt="question mark cursor" /></td>
</tr>
<tr>
<td>VAL_HOUR_GLASS_CURSOR</td>
<td><img src="image8" alt="hourglass cursor" /></td>
</tr>
<tr>
<td>VAL_HIDDEN_CURSOR</td>
<td><img src="image9" alt="hidden cursor" /></td>
</tr>
<tr>
<td>VAL_SIZE_NS_CURSOR</td>
<td><img src="image10" alt="size NS cursor" /></td>
</tr>
<tr>
<td>VAL_SIZE_EW_CURSOR</td>
<td><img src="image11" alt="size EW cursor" /></td>
</tr>
<tr>
<td>VAL_SIZE_NW_SE_CURSOR</td>
<td><img src="image12" alt="size NW-SE cursor" /></td>
</tr>
<tr>
<td>VAL_SIZE_NE_SW_CURSOR</td>
<td><img src="image13" alt="size NE-SW cursor" /></td>
</tr>
</tbody>
</table>
Table 3-5 lists valid font values.

Table 3-5. Font Values

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform independent fonts</strong></td>
<td></td>
</tr>
<tr>
<td>Platform independent fonts</td>
<td>VAL_MENU_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_MESSAGE_BOX_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_DIALOG_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_EDITOR_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_APP_FONT</td>
</tr>
<tr>
<td><strong>Platform independent metafonts</strong></td>
<td></td>
</tr>
<tr>
<td>Platform independent metafonts</td>
<td>VAL_MENU_META_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_MESSAGE_BOX_META_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_DIALOG_META_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_EDITOR_META_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_APP_META_FONT</td>
</tr>
<tr>
<td><strong>LabWindows/CVI supplied metafonts</strong></td>
<td></td>
</tr>
<tr>
<td>LabWindows/CVI supplied metafonts</td>
<td>VAL_7SEG_META_FONT</td>
</tr>
<tr>
<td></td>
<td>VAL_SYSTEM_META_FONT</td>
</tr>
<tr>
<td><strong>Host fonts</strong></td>
<td></td>
</tr>
<tr>
<td>Host fonts</td>
<td>See discussion following this table.</td>
</tr>
<tr>
<td><strong>User-defined metafonts</strong></td>
<td></td>
</tr>
<tr>
<td>User-defined metafonts</td>
<td>See discussion following this table.</td>
</tr>
</tbody>
</table>

**Platform Independent Fonts That Are Resident on PCs and UNIX**

These fonts contain typeface information only and use typefaces native to each platform supported by LabWindows/CVI. LabWindows/CVI uses VAL_MESSAGE_BOX_META_FONT for message boxes. Under Windows 95, this font is for message boxes. On other platforms, this font is the same as VAL_DIALOG_META_FONT.

**Platform-Independent Metafonts That Are Resident on PCs and UNIX**

Metafonts contain typeface information, point size, and text styles such as bold, underline, italic, and strikeout. The platform independent metafonts listed in Table 3-5 use typefaces that are native to each platform. They have been designed so that the height and width of the font is as similar as possible across platforms. These metafonts are used in the LabWindows/CVI environment. VAL_MENU_META_FONT is bold and is used for menu and menu item names. VAL_DIALOG_META_FONT is bold and is used for labels in dialog boxes and function panels.
**Metafonts Supplied by LabWindows/CVI**

These metafonts are included in LabWindows/CVI, but use typefaces that are not native to PC or UNIX systems. VAL_7SEG_META_FONT provides compatibility with LabWindows for DOS. VAL_SYSTEM_META_FONT provides compatibility with function panel help text and user interface panels from LabWindows for DOS that use extended IBM PC characters. Although the extended IBM characters are provided, their use is discouraged since other fonts do not contain them. (Cut/Paste operations with extended IBM characters fail across applications that use other fonts.)

**Host Fonts**

In addition to the fonts provided with LabWindows/CVI, you can use any host font supported on your system. For example, Arial, Courier, and Roman are available under Windows.

**Note:** When you change the font and point size of text, always set the font before setting the point size or style attributes.

**User Defined Metafont**

You can create metafonts with the CreateMetaFont function. You can then apply the metafont to any control or panel attribute that takes a font value. The typeface, point size, and text styles defined by the metafont are all set at once. In addition, the PlotText and GetTextDisplaySize functions require a metafont as an input parameter.

**Programming with Menu Bars**

This section describes how to use User Interface Library functions to control the elements of user interface menu bars.

**Menu Bar Functions**

Use LoadMenuBar to load a menu bar from a resource file created in the User Interface Editor and display it on a panel. When LoadMenuBar loads a menu bar from a resource file, it references the menu bar using the constant name that you assigned to the menu bar in the User Interface Editor. LoadMenuBar returns a handle that you use in subsequent User Interface Library functions to reference the menu bar. After a menu bar has been loaded, it generates a user event whenever the user selects a menu bar command. Refer to the Processing Menu Bar Events section of this chapter for additional information.
Use **NewMenuBar** to create a new menu bar during program execution. **NewMenuBar** returns a handle that you use to reference the menu bar in subsequent operations. Use the single parameter of **NewMenuBar** to specify the panel on which you want to locate the menu bar. Use **NewMenu**, **NewSubMenu**, and **NewMenuItem** to construct the pull-down menu system.

Use **SetPanelMenuBar** to assign a menu bar to a panel. You can assign a single panel only one menu bar at a time. Multiple panels can share the same menu bar.

Use **GetPanelMenuBar** to get the menu bar handle associated with a panel.

Use **GetSharedMenuBarEventPanel** to get the handle of the panel on which a menu bar event occurred when multiple panels share a menu bar. You need this function only when using **GetUserEvent**; the panel handle is passed as a parameter to a menu callback function.

Use **GetMenuBarAttribute** to obtain a particular menu bar or menu item attribute.

Use **SetMenuBarAttribute** to set a particular menu bar or menu item attribute.

Use **InsertSeparator** to programmatically place a dividing line between menu items.

Use **NewMenu**/**DiscardMenu**, **NewMenuItem**/**DiscardMenuItem**, and **NewSubMenu**/**DiscardSubMenu** to programmatically alter a menu system.

Use **EmptyMenu** to remove the contents of a particular menu or sub-menu.

Use **EmptyMenuBar** to remove the menu bar from the screen and remove its contents from memory but retain the menu bar handle.

Use **DiscardMenuBar** to remove the menu bar from the screen and free its handle from memory.

**Processing Menu Bar Events**

There are two ways to process menu bar events. One way is to assign callback functions to menu items and immediate action menus. When a commit event is generated, the appropriate callback function executes. Alternatively you can use an event loop that includes a call to **GetUserEvent**. When a commit event is generated, **GetUserEvent** returns the appropriate menu bar handle and menu item ID, and the program conditionally executes portions of code. Commit events are generated when the user selects a menu item or immediate action menu item.
Using Callback Functions...

If you create your menu bar in the User Interface Editor, you can assign callback function names to menu items and immediate action menus from within the editor. When `LoadMenuBar` loads the menu bar, LabWindows/CVI automatically installs your callback functions and calls them whenever commit events are generated on menu items and immediate action menus.

If you create your menu bar programmatically via `NewMenuBar`, you can use `InstallMenuCallback` to install callback functions for immediate action menus. Menu item callbacks are installed as a parameter to `NewMenuItem`. Your callback functions are then called whenever commit events are generated on menu items or immediate action menus. You should use the `MenuCallbackPtr` typedef from `userint.h` as a model to declare your menu callback functions in your program.

You can also use `InstallMenuDimmerCallback` to install a callback function that is called just before a pull-down menu appears when the user clicks a menu bar or presses a menu short-cut key. This callback function can update the state of dimmed menu items before the pull-down menu is displayed to the user. You should use the `MenuDimmerCallbackPtr` typedef from `userint.h` as a model to declare this callback function in your program.

Callbacks must be initiated through a call to `RunUserInterface` or a `GetUserEvent` loop.

Using an Event Loop...

When the user generates a commit event on a menu item or immediate action menu, the `GetUserEvent` function returns the appropriate menu bar handle and menu ID as parameters. `GetUserEvent` can operate in one of two ways.

- `GetUserEvent` waits for the user to generate an event before returning to the calling program.
- `GetUserEvent` returns immediately whether or not an event has occurred. If a menu event occurred, `GetUserEvent` returns a valid menu or menu item ID.
Menu Bar Attributes

Table 3-6 lists menu and menu item attributes that you can retrieve or change through `GetMenuBarAttribute` and `SetMenuBarAttribute`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Menu and Menu Item Attribute**  | **ATTR_DIMMED**         | int          | 1 = menu/item is disabled  
0 = menu/item is enabled  
Pass 0 (zero) if the attribute corresponds to the entire menu bar.                                                                                 |
| **ATTR_CALLBACK_DATA**            | void *                  |              | A pointer to user data area that will be passed to the menu or menu item callback function.                                                  |
| **ATTR_CALLBACK_FUNCTION_POINTER**| void *                  |              | A pointer to the callback function for the menu or menu item.                                                                                  |
| **ATTR_CALLBACK_NAME**            | char *                  |              | The name of the callback function associated with the menu or menu item (`GetMenuBarAttribute` only).                                         |
| **ATTR_CALLBACK_NAME_LENGTH**     | int                     |              | The number of characters in the name of the menu or menu item callback name (`GetMenuBarAttribute` only).                                      |
| **ATTR_CONSTANT_NAME**            | char *                  |              | The constant name assigned to the menu or menu item in the .uir file when created by the user interface editor (`GetMenuBarAttribute` only). |
| **ATTR_CONSTANT_NAME_LENGTH**     | int                     |              | The number of characters in the constant name of the menu or menu item (`GetMenuBarAttribute` only).                                           |
Table 3-6. Menu and Menu Item Attributes (Continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Menu</strong></td>
<td>ATTR_MENU_NAME</td>
<td>char *</td>
<td>The name of the menu item.</td>
</tr>
<tr>
<td></td>
<td>ATTR_MENU_NAME_LENGTH</td>
<td>int</td>
<td>The number of characters in the menu item (GetMenuBarAttribute only).</td>
</tr>
<tr>
<td><strong>Menu Bar</strong></td>
<td>ATTR_DRAW_LIGHT_BEVEL</td>
<td>int</td>
<td>Indicates whether the menubar draws with a light or dark bevel at the bottom.</td>
</tr>
<tr>
<td>Only</td>
<td></td>
<td></td>
<td>(Applies only to Windows 95.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Dark Bevel (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = Light Bevel</td>
</tr>
<tr>
<td></td>
<td>ATTR_NUM_MENUS</td>
<td>int</td>
<td>The number of menus for the menubar.</td>
</tr>
<tr>
<td><strong>Menu Item</strong></td>
<td>ATTR_CHECKED</td>
<td>int</td>
<td>1 = a check mark by the menu item</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = no check mark by the menu item</td>
</tr>
<tr>
<td></td>
<td>ATTR_IS_SEPARATOR</td>
<td>int</td>
<td>1 = the menu item is a separator,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = the menu item is not a separator (GetMenuBarAttribute only).</td>
</tr>
<tr>
<td></td>
<td>ATTR_ITEM_NAME</td>
<td>char *</td>
<td>The name of the menu item.</td>
</tr>
<tr>
<td></td>
<td>ATTR_ITEM_NAME_LENGTH</td>
<td>int</td>
<td>The number of characters in the menu item name (GetMenuBarAttribute only).</td>
</tr>
<tr>
<td></td>
<td>ATTR_NUM_MENU_ITEMS</td>
<td>int</td>
<td>The number of menu items for the menu.</td>
</tr>
<tr>
<td></td>
<td>ATTR_SHORTCUT_KEY</td>
<td>int</td>
<td>Key code for the menu item shortcut key (see Table 3-7)</td>
</tr>
<tr>
<td></td>
<td>ATTR_SUBMENU_ID</td>
<td>int</td>
<td>The constant ID for the submenu (GetMenuBarAttribute only).</td>
</tr>
</tbody>
</table>
Menu Bar Attribute Discussion

In source code, a shortcut key is represented by a 4-byte integer consisting of three bit fields, 0×00MMVVAA, where:

MM = the modifier key
VV = the virtual key
AA = the ASCII key

When you construct a shortcut key, a modifier key is bit-wise OR’ed with a virtual key or an ASCII key. Either the ASCII field or the virtual key field will be all zeros. For example, VAL_SHIFT_MODIFIER | VAL_F1_VKEY produces a shortcut key of <Shift-F1>, and VAL_MENUKEY_MODIFIER | 'D' produces a shortcut key of <Ctrl-D>.

Virtual keys do not require modifiers but ASCII keys require at least one modifier.

Table 3-7 shows the modifiers and virtual keys for shortcut keys.

Table 3-7. Key Modifiers and Virtual Keys

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key modifiers</td>
<td>VAL_SHIFT_MODIFIER</td>
</tr>
<tr>
<td></td>
<td>VAL_SHIFT_AND_MENUKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_MENUKEY_MODIFIER</td>
</tr>
<tr>
<td></td>
<td>&lt;Ctrl&gt; on the PC and &lt;Ctrl&gt; or &lt;Meta&gt; on the SPARCstation.</td>
</tr>
<tr>
<td></td>
<td>VAL_UNDERLINE_MODIFIER</td>
</tr>
<tr>
<td></td>
<td>&lt;Alt&gt; on the PC and the SPARCstation.</td>
</tr>
<tr>
<td></td>
<td>Also, optionally, &lt;META&gt; on the SPARCstation.</td>
</tr>
</tbody>
</table>

(continues)
Table 3-7. Key Modifiers and Virtual Keys (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual key codes</td>
<td>VAL_FWD_DELETE_VKEY not available on SPARCstation keyboards.</td>
</tr>
<tr>
<td></td>
<td>VAL_BACKSPACE_VKEY &lt;del&gt; and &lt;Back Space&gt; on the SPARCstation.</td>
</tr>
<tr>
<td></td>
<td>VAL_ESC_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_TAB_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_ENTER_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_UP_ARROW_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_DOWN_ARROW_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_LEFT_ARROW_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_RIGHT_ARROW_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_INSERT_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_HOME_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_END_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_PAGE_UP_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_PAGE_DOWN_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F1_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F2_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F3_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F4_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F5_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F6_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F7_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F8_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F9_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F10_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F11_VKEY</td>
</tr>
<tr>
<td></td>
<td>VAL_F12_VKEY</td>
</tr>
</tbody>
</table>

When you use `GetMenuBarAttribute`, the three constants shown in Table 3-8 are available to mask the three bit fields of a shortcut key.
Table 3-8. Constants for Masking Three Bit Fields in GetMenuBarAttribute

<table>
<thead>
<tr>
<th>Value</th>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_MODIFIER_MASK</td>
<td>0x00FF0000</td>
</tr>
<tr>
<td>VAL_VKEY_MASK</td>
<td>0x0000FF00</td>
</tr>
<tr>
<td>VAL_ASCII_KEY_MASK</td>
<td>0x000000FF</td>
</tr>
</tbody>
</table>

To separate each field of a key code, perform a bit-wise AND operation on the key code using each of the three masks.

Programming with Controls

This section describes how to use the User Interface Library functions to control the elements of user interface controls.

Control Functions for All Controls

When you use LoadPanel to load a panel from a resource file created in the user interface editor into memory, it also loads the controls on that panel. Use the defined constant names (IDs) assigned to the controls on the panel in the User Interface Editor to reference the controls in the control functions.

Use NewCtrl to create a new control during program execution. NewCtrl returns an ID you use to reference the control in subsequent operations. Use the first parameter of NewCtrl to specify the panel on which the control is to appear. The second parameter of NewCtrl specifies the control style. (Table 3-10 lists control styles.) You also specify the name and position of the control through parameters to NewCtrl.

Use DuplicateCtrl to create a new control that is a duplicate of a control that was loaded by LoadPanel or created by NewCtrl. DuplicateCtrl returns an ID that you use to reference the control in subsequent operations. You specify the destination panel, name, and position of the control through parameters to DuplicateCtrl.

Use SetCtrlVal to set a control to a particular value. This function is not valid for Graph and Strip Chart controls.

Use GetCtrlVal to obtain the current value of a control. This function is not valid for Graph and Strip Chart controls.

Use GetActiveCtrl to obtain the control that receives keyboard events when its panel is the active panel.
Use `SetActiveCtrl` to establish the control that receives keyboard events when its panel is the active panel. The label of the active control is inset when its panel is the active panel.

Use `DefaultCtrl` to restore a control to its default value. If the control is visible, the program updates it to reflect its default value. You assign default values to controls in the User Interface Editor or through `SetCtrlAttribute`. *This function is not valid for Graph and Strip Chart controls.*

Use `GetCtrlBoundingRect` to obtain the top, left, height, and width of the rectangle bounding the control and its label.

Use `GetCtrlAttribute` to obtain a particular attribute of a control.

Use `SetCtrlAttribute` to set a particular attribute of a control. When changing control attributes that affect the font of a control, `ATTR_TEXT_FONT` should be the first attribute you modify.

Use `DiscardCtrl` to remove a control from memory. If its panel is visible, the control is removed from the screen. LabWindows/CVI allows you to call `DiscardCtrl` only from an `EVENT_COMMIT` event. A call to `DiscardCtrl` from any other type of event may cause unpredictable behavior.

**Control Functions for List Controls (List Boxes and Rings)**

List controls are unique because they contain an indexed set of label/value pairs. Labels are strings that appear on the ring, and values can be of any type. For more information on a special type of ring control, the picture ring control, see the *Programming Picture Controls* section.

**List Boxes and Rings...**

Use `InsertListItem` to add an item to the list.

Use `DeleteListItem` to delete one or more items from the list.

Use `ReplaceListItem` to replace a given list item with a new item. To preserve the existing label and change only the value, pass "0" as the label parameter.

Use `GetCtrlVal` to obtain the value of the currently active list item.

Use `SetCtrlVal` to set the currently active list item based on a given value.

Use `GetCtrlIndex` to obtain the zero-based index of the currently active list item.

Use `SetCtrlIndex` to set the currently active list item based on a given zero-based index.
Use GetValueFromIndex to obtain the value of a list control corresponding to a given zero-based index of the list.

Use GetValueLengthFromIndex to obtain the length of the value (strings only) of a list control corresponding to a given zero-based index of the list.

Use GetIndexFromValue to obtain a zero-based index of a list control corresponding to a given value.

Use GetNumListItems to obtain the number of items contained in a list control.

Use GetLabelFromIndex to obtain the label of a list item given a zero-based index.

Use GetLabelLengthFromIndex to obtain the length of the label of a list item given a zero-based index.

Use ClearListCtrl to clear the contents of a list control.

**List Boxes only...**

Use SetListItemImage to set the image associated with a particular list item. (List item images are the folder icons that appear in the Open/Save File dialog box in LabWindows/CVI.)

Use GetListItemImage to obtain the image associated with a particular list item. (List item images are the folder icons that appear in the Open/Save File dialog box in LabWindows/CVI.)

Use IsListItemChecked to determine if a particular list item is checked.

Use CheckListItem to programmatically check a list item.

Use GetNumCheckedItems to determine the number of checked list items.

**Control Functions for Text Boxes**

Text boxes in the User Interface library can contain a finite number of lines. If the number of lines multiplied by the pixel height of the font exceeds 32767, the text box does not scroll. If you use NIDialogMetaFont or the NIEditorMetaFont, your text box can contain a maximum of approximately 2500 lines.

Use GetCtrlVal to obtain the text in the text box.

Use SetCtrlVal to append text to a text box.

Use InsertTextBoxLine to add text into a text box starting at a given line.

Use DeleteTextBoxLine to delete one or more lines from a text box.
Use `ReplaceTextBoxLine` to replace a given line with new text.

Use `GetNumTextBoxLines` to obtain the number of lines used in a text box.

Use `GetTextBoxLineLength` to obtain the length of a particular line in a text box.

Use `GetTextBoxLine` to obtain parameters a particular line in a text box.

Use `ResetTextBox` to replace the contents of a text box.

**Processing Control Events**

Once a panel is displayed, there are two ways to process events from the controls on that panel. You can assign callback functions to the controls. When any type of event occurs on a control, the appropriate callback function executes. Alternatively, you can use an event loop that includes a call to `GetUserEvent`. When the user generates a commit event on a control, `GetUserEvent` returns the appropriate control ID, and your program can conditionally execute portions of code. Commit events occur on a control when a user changes the value on a control and presses <Enter> or <Tab> or clicks on another control with the mouse.

**Using Callback Functions...**

If you create your controls in the User Interface Editor, you can assign callback function names to the controls from within the editor. When `LoadPanel` loads the panel, LabWindows/CVI automatically installs your callback functions and calls them whenever events are generated on the controls.

If you create your controls programmatically via `NewCtrl`, you can use `InstallCtrlCallback` to assign callback function names to the controls. Your callback function is then called whenever an event is generated on the panel. You should use the `CtrlCallbackPtr` typedef from `userint.h` as a model to declare your control callback functions in your program.

Callbacks must be initiated through a call to `RunUserInterface` or to a `GetUserEvent` loop.

**Note:** Do not call `DiscardCtrl` from within the callback function of the control you want to discard. Doing this gives unpredictable results.

**Using an Event Loop...**

When the user generates a commit event on a control, the `GetUserEvent` function returns the appropriate ID of the control as a parameter. `GetUserEvent` can operate in one of two ways.
• `GetUserEvent` waits for the user to generate an event before returning to the calling program.

• `GetUserEvent` returns immediately whether or not an event occurs. If no event occurs, the control ID parameter is -1.

### Control Attributes

Table 3-9 lists control attributes that you can retrieve or change through `GetCtrlAttribute` and `SetCtrlAttribute`.

#### Table 3-9. Control Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CALLBACK_DATA</td>
<td>void *</td>
<td>A pointer to user data that is passed to the panel callback function</td>
</tr>
<tr>
<td>ATTR_CALLBACK_FUNCTION_POINT</td>
<td>void *</td>
<td>A pointer to the callback function for the control.</td>
</tr>
<tr>
<td>ATTR_CALLBACK_NAME</td>
<td>char *</td>
<td>The name of the callback function associated with the control (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_CALLBACK_NAME_LENGTH</td>
<td>int</td>
<td>The number of characters in the name of the control callback name (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_CONSTANT_NAME</td>
<td>char *</td>
<td>The constant name assigned to the control in the .uir file when created by the User Interface Editor (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_CONSTANT_NAME_LENGTH</td>
<td>int</td>
<td>The number of characters in the constant name of the control (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_CTRL_STYLE</td>
<td>int</td>
<td>The control style, see Table 3-10 (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_DIMMED</td>
<td>int</td>
<td>1 = control disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = control enabled</td>
</tr>
</tbody>
</table>

(continues)
### For all controls (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_LEFT</td>
<td>int</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>ATTR_NEXT_CTRL</td>
<td>int</td>
<td>The next control on the panel (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_OVERLAPPED</td>
<td>int</td>
<td>Indicates whether the control is overlapped by another control (or its own parts). (GetCtrlAttribute only)</td>
</tr>
<tr>
<td>ATTR_TOP</td>
<td>int</td>
<td>-32768 to 32767</td>
</tr>
</tbody>
</table>
| ATTR_VISIBLE       | int   | 1 = control visible
0 = control invisible |
| ATTR_WIDTH         | int   | 0 to 32767                                                                 |
| ATTR_ZPLANE_POSITION | int  | The order of the control in the z-plane. The lowest ordered control (0) is on top. |

### For all controls except simple strings, simple numerics, and simple rings

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_HEIGHT</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
</tbody>
</table>

### For all controls except indicator-only controls (indicator-only controls are: decorations, strip charts, text messages, timers, and pictures)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CTRL_MODE</td>
<td>int</td>
<td>VAL_HOT or VAL_NORMAL or VAL_VALIDATE or VAL_INDICATOR</td>
</tr>
<tr>
<td>ATTR_CTRL_TAB_POSITION</td>
<td>int</td>
<td>The tab order of the control.</td>
</tr>
</tbody>
</table>

### For all controls except decorations, canvases, graphs and strip charts

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CTRL_VAL</td>
<td>same as control type</td>
<td>Same functionality as GetCtrlVal and SetCtrlVal.</td>
</tr>
</tbody>
</table>
For all controls with frames (this includes text boxes, strings, numerics, simple rings, ring meters, ring gauges, list boxes, decorations, and pictures)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_FRAME_COLOR</td>
<td>int</td>
<td>RGB value—see discussion following this table.</td>
</tr>
</tbody>
</table>

For controls with labels (all controls except decorations, and text messages)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ATTR_LABEL_BOLD       | int        | 1 = label is bold  
                          0 = label is not bold                                                      |
| ATTR_LABEL_COLOR      | int        | RGB value—see discussion following this table.                              |
| ATTR_LABEL_FONT       | char *     | The font of the control label (see Table 3-5).                              |
| ATTR_LABEL_FONT_NAME_LENGTH | int    | Number of characters in the name of the label font (GetCtrlAttribute only). |
| ATTR_LABEL_ITALIC     | int        | 1 = label in italics  
                          0 = label not in italics                                                    |
| ATTR_LABEL_POINT_SIZE | int        | Point size of the label; range = 1 to 32767.                                 |
| ATTR_LABEL_STRIKEOUT  | int        | 1 = label has strikeout  
                          0 = label does not have strikeout                                           |
| ATTR_LABEL_TEXT       | char *     | The label of the control.                                                  |
| ATTR_LABEL_TEXT_LENGTH | int     | Number of characters in the label (GetCtrlAttribute only).                 |
| ATTR_LABEL_UNDERLINE  | int        | 1 = label underlined  
                          0 = label not underlined                                                    |
| ATTR_LABEL_VISIBLE    | int        | 1 = label is visible  
                          0 = label is invisible                                                      |
### For controls with labels except non-picture command buttons

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_LABEL_BGCOLOR</td>
<td>int</td>
<td>Label background color, RGB value.</td>
</tr>
<tr>
<td>ATTR_LABEL_HEIGHT</td>
<td>int</td>
<td>Height of label; range = 0 to 32767.</td>
</tr>
<tr>
<td>ATTR_LABEL_JUSTIFY</td>
<td>int</td>
<td>VAL_LEFT_JUSTIFIED or VAL_RIGHT_JUSTIFIED or VAL_CENTER_JUSTIFIED</td>
</tr>
<tr>
<td>ATTR_LABEL_LEFT</td>
<td>int</td>
<td>Left position of label, range = -32768 to 32767 or use VAL_AUTO_CENTER.</td>
</tr>
<tr>
<td>ATTR_LABEL_RAISED</td>
<td>int</td>
<td>1 = label is raised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = label is not raised</td>
</tr>
<tr>
<td>ATTR_LABEL_SIZE_TO_TEXT</td>
<td>int</td>
<td>1 = label border restricted to label text size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = label border not restricted to label text size</td>
</tr>
<tr>
<td>ATTR_LABEL_TOP</td>
<td>int</td>
<td>Top position of label, range = -32768 to 32767 or use VAL_AUTO_CENTER or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VAL_RIGHT_ANCHOR or VAL_LEFT_ANCHOR or</td>
</tr>
<tr>
<td>ATTR_LABEL_WIDTH</td>
<td>int</td>
<td>Width of label, range = 0 to 32767.</td>
</tr>
</tbody>
</table>

### For controls with text except graphs and strip charts (controls with text are all controls except: decorations, pictures, canvases, LEDs, and buttons without on/off text)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_TEXT_BOLD</td>
<td>int</td>
<td>1 = text is bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = text is not bold</td>
</tr>
<tr>
<td>ATTR_TEXT_COLOR</td>
<td>int</td>
<td>RGB value—see discussion below.</td>
</tr>
<tr>
<td>ATTR_TEXT_FONT</td>
<td>char *</td>
<td>The font of the text (see Table 3-5).</td>
</tr>
</tbody>
</table>
For controls with text except graphs and strip charts (controls with text are all controls except: decorations, pictures, canvases, LEDs, and buttons without on/off text)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_TEXT_FONT_NAME_LENGTH</td>
<td>int</td>
<td>Number of characters in the name of the text font (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_TEXT_ITALIC</td>
<td>int</td>
<td>1 = text in italics 0 = text not in italics</td>
</tr>
<tr>
<td>ATTR_TEXT_POINT_SIZE</td>
<td>int</td>
<td>Point size of the text, range = 1 to 32767.</td>
</tr>
<tr>
<td>ATTR_TEXT_STRIKEOUT</td>
<td>int</td>
<td>1 = text has strikeout 0 = text does not have strikeout</td>
</tr>
<tr>
<td>ATTR_TEXT_UNDERLINE</td>
<td>int</td>
<td>1 = text underlined 0 = text not underlined</td>
</tr>
</tbody>
</table>

For controls with text except graphs, strip charts, ring slides, binary switches, and text buttons

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_TEXT_BGCOLOR</td>
<td>int</td>
<td>RGB value—see discussion below.</td>
</tr>
</tbody>
</table>

For controls with text except graphs, strip charts, ring slides, pop-up rings, binary switches, and text buttons

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_TEXT_JUSTIFY</td>
<td>int</td>
<td>VAL_LEFT_JUSTIFIED or VAL_RIGHT_JUSTIFIED or VAL_CENTER_JUSTIFIED</td>
</tr>
</tbody>
</table>

For controls with variable data types (this includes numerics, rings, binary switches, and list boxes)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_DATA_TYPE</td>
<td>int</td>
<td>See Table 3-11.</td>
</tr>
</tbody>
</table>
For label/value controls (slides, rings, binary switches, and list boxes)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CTRL_INDEX</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
<tr>
<td>ATTR_DFLT_INDEX</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
</tbody>
</table>

For numerics

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CHECK_RANGE</td>
<td>int</td>
<td>VAL_COERCER or VAL_IGNORE or VAL_NOTIFY (see Table 3-13).</td>
</tr>
<tr>
<td>ATTR_DFLT_VALUE</td>
<td>same as control type</td>
<td>Default value of the control.</td>
</tr>
<tr>
<td>ATTR_FORMAT</td>
<td>int</td>
<td>See Table 3-12.</td>
</tr>
<tr>
<td>ATTR_INCR_VALUE</td>
<td>same as control type</td>
<td>Increments for INC/DEC arrows of the control.</td>
</tr>
<tr>
<td>ATTR_MAX_VALUE</td>
<td>same as control type</td>
<td>Maximum value of the control.</td>
</tr>
<tr>
<td>ATTR_MIN_VALUE</td>
<td>same as control type</td>
<td>Minimum value of the control.</td>
</tr>
<tr>
<td>ATTR_PRECISION</td>
<td>int</td>
<td>The numeric precision (0 to 15).</td>
</tr>
<tr>
<td>ATTR_NSCROLL_OFFSET_MAX</td>
<td>int</td>
<td>The maximum horizontal scroll bar offset in pixels; range 0 to 32767.</td>
</tr>
</tbody>
</table>
| ATTR_SHOW_RADIX     | int    | 1 = radix is shown
0 = radix is not shown |

For picture and slide rings and numerics

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ATTR_SHOW_INCDEC_ARROWS | int    | 1 = INC/DEC arrows are shown
0 = INC/DEC arrows are not shown |
### For strings and text boxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_MAX_ENTRY_LENGTH</td>
<td>int</td>
<td>Maximum # of characters; -1 means no limit.</td>
</tr>
<tr>
<td>ATTR_TEXT_SELECTION_LENGTH</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
<tr>
<td>ATTR_TEXT_SELECTION_START</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
</tbody>
</table>

### For text messages, strings, and text boxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_STRING_TEXT_LENGTH</td>
<td>int</td>
<td>0 to 32767 (GetCtrlAttribute only).</td>
</tr>
</tbody>
</table>

### For text boxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ENTER_IS_NEWLINE</td>
<td>int</td>
<td>The &lt;Enter&gt; key causes a newline, otherwise a &lt;Ctrl-Enter&gt; is used. 1 = &lt;Enter&gt; for newline. 0 = &lt;Ctrl-Enter&gt; for newline.</td>
</tr>
<tr>
<td>ATTR_EXTRA_LINES</td>
<td>int</td>
<td>Number of lines retained off-screen (0 to 32767); -1 = unlimited number of lines retained off screen</td>
</tr>
<tr>
<td>ATTR_HSCROLL_OFFSET</td>
<td>int</td>
<td>Horizontal scroll offset; range = 0 to 32767 pixels.</td>
</tr>
<tr>
<td>ATTR_TOTAL_LINES</td>
<td>int</td>
<td>Number of lines used (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_WRAP_MODE</td>
<td>int</td>
<td>VAL_CHAR_WRAP or VAL_LINE_WRAP or VAL_WORD_WRAP</td>
</tr>
</tbody>
</table>
For text boxes and list boxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_FIRST_VISIBLE_LINE</td>
<td>int</td>
<td>Index of first visible line (zero-based).</td>
</tr>
<tr>
<td>ATTR_SCROLL_BAR_COLOR</td>
<td>int</td>
<td>Any valid RGB value.</td>
</tr>
<tr>
<td>ATTR_SCROLL_BAR_SIZE</td>
<td>int</td>
<td>VAL_SMALL_SCROLL_BARS or VAL_LARGE_SCROLL_BARS</td>
</tr>
<tr>
<td>ATTR_SCROLL_BARS</td>
<td>int</td>
<td>VAL_NO_SCROLL_BARS or VAL_HORIZ_SCROLL_BAR or VAL_VERT_SCROLL_BAR or VAL_BOTH_SCROLL_BARS (Horizontal scroll bars not valid for list boxes.)</td>
</tr>
<tr>
<td>ATTR_VISIBLE_LINES</td>
<td>int</td>
<td>Number of visible lines (0 to 32767).</td>
</tr>
<tr>
<td>ATTR_TEXT_CLICK_Toggles_CHECK</td>
<td>int</td>
<td>1 = clicking on the text area toggle the checkmark 0 = clicking on the text area does not toggle the checkmark</td>
</tr>
</tbody>
</table>

For list boxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ALLOW_ROOM_FOR_IMAGES</td>
<td>int</td>
<td>Specifies whether, when calculating the list box height, to assume that one or more list box labels may contain an image. Normally, the calculation of the list box height takes into account the image height only if there currently is an image in the list box. To make sure that the list box height always takes into account the height of an image, set this attribute to TRUE (1).</td>
</tr>
<tr>
<td>ATTR_CHECK_MODE</td>
<td>int</td>
<td>1 = enable checking 0 = disable checking</td>
</tr>
</tbody>
</table>
### For list boxes (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CHECK_STYLE</td>
<td>int</td>
<td>VAL_CHECK_MARK or VAL_CHECK_BOX</td>
</tr>
<tr>
<td>ATTR_HILITE_CURRENT_ITEM</td>
<td>int</td>
<td>Specifies whether to highlight the currently selected item in a list box. (The highlight is shown in reversed colors when list box is active, a dashed box when inactive.)</td>
</tr>
<tr>
<td>ATTR_TEXT_CLICK_TOGGLE_CHECK</td>
<td>int</td>
<td>1 = clicking on the text area toggles the check mark  0 = clicking on the text area does not toggle the check mark</td>
</tr>
</tbody>
</table>

### For strings, numerics, and text boxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_NO_EDIT_TEXT</td>
<td>int</td>
<td>1 = enable text editing  0 = disable text editing</td>
</tr>
</tbody>
</table>

### For text messages

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_TEXT_RAISED</td>
<td>int</td>
<td>1 = text is raised  0 = text is not raised</td>
</tr>
<tr>
<td>ATTR_SIZE_TO_TEXT</td>
<td>int</td>
<td>1 = message border restricted to text size  0 = message border not restricted to text size</td>
</tr>
</tbody>
</table>

### For command buttons

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CMD_BUTTON_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_SHORTCUT_KEY</td>
<td>int</td>
<td>See discussion of shortcut keys in Menu Bar Attributes section of this chapter.</td>
</tr>
</tbody>
</table>
### For binary switches

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_BINARY_SWITCH_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_OFF_VALUE</td>
<td>same as control type</td>
<td>Value of control when OFF.</td>
</tr>
<tr>
<td>ATTR_OFF_VALUE_LENGTH</td>
<td>int</td>
<td>Only if string value (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_ON_VALUE</td>
<td>same as control type</td>
<td>Value of control when ON.</td>
</tr>
<tr>
<td>ATTR_ON_VALUE_LENGTH</td>
<td>int</td>
<td>Only if string value (GetCtrlAttribute only).</td>
</tr>
</tbody>
</table>

### For LEDs and buttons except command buttons

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_OFF_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_ON_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
</tbody>
</table>

### For text buttons and binary switches

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_OFF_TEXT</td>
<td>char *</td>
<td>Text displayed in OFF position.</td>
</tr>
<tr>
<td>ATTR_OFF_TEXT_LENGTH</td>
<td>int</td>
<td>Number of characters in the OFF text (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_ON_TEXT</td>
<td>char *</td>
<td>Text displayed in ON position.</td>
</tr>
<tr>
<td>ATTR_ON_TEXT_LENGTH</td>
<td>int</td>
<td>Number of characters in the ON text (GetCtrlAttribute only).</td>
</tr>
</tbody>
</table>
For numerics with digital displays (slides, knobs, dials, meters, and gauges)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_DIG_DISP_LEFT</td>
<td>int</td>
<td>Left position of the digital display; range = -32768 to 32767.</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_TOP</td>
<td>int</td>
<td>Top position of the digital display; range = -32768 to 32767.</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_HEIGHT</td>
<td>int</td>
<td>Height of the digital display, range 0 to 32767.</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_WIDTH</td>
<td>int</td>
<td>Width of the digital display; range = 0 to 32767.</td>
</tr>
<tr>
<td>ATTR_SHOW_DIG_DISP</td>
<td>int</td>
<td>1 = show digital display 0 = hide digital display</td>
</tr>
</tbody>
</table>

For numerics and ring slides, knobs, dials, meters, and gauges

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_FILL_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_MARKER_STYLE</td>
<td>int</td>
<td>VAL_NO_MARKERS or VAL_NO_INNER_MARKERS or VAL_FULL_MARKERS</td>
</tr>
<tr>
<td>ATTR_NEEDLE_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_SLIDER_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_TICK_STYLE</td>
<td>int</td>
<td>VAL_NO_TICKS or VAL_NO_MINOR_TICKS or VAL_FULL_TICKS</td>
</tr>
</tbody>
</table>
### For numeric and ring slides

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_FILL_HOUSING_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_FILL_OPTION</td>
<td>int</td>
<td>VAL_NO_FILL, VAL_TOP_FILL, VAL_BOTTOM_FILL, VAL_RIGHT_FILL, or VAL_LEFT_FILL</td>
</tr>
<tr>
<td>ATTR_SLIDER_HEIGHT</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
<tr>
<td>ATTR_SLIDER_WIDTH</td>
<td>int</td>
<td>0 to 32767</td>
</tr>
</tbody>
</table>

### For numeric and ring knobs, dials, and gauges

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_MARKER_END_ANGLE</td>
<td>int</td>
<td>Range = 0 to 359</td>
</tr>
<tr>
<td>ATTR_MARKER_START_ANGLE</td>
<td>int</td>
<td>Range = 0 to 359</td>
</tr>
</tbody>
</table>

### For color numerics

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_SHOW_MORE_BUTTON</td>
<td>int</td>
<td>1/0 = show/do not show the MORE button on the color numeric pop-up.</td>
</tr>
<tr>
<td>ATTR_SHOW_TRANSPARENT</td>
<td>int</td>
<td>1/0 = show/do not show the transparent icon on the color numeric pop-up.</td>
</tr>
</tbody>
</table>

### For menu rings

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_MENU_ARROW_COLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table.</td>
</tr>
</tbody>
</table>
### For timer controls

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_INTERVAL</td>
<td>double</td>
<td>The interval at which the timer control callback function will be called (in seconds).</td>
</tr>
</tbody>
</table>
| ATTR_ENABLED     | int      | 0 = Timer callback is disabled  
|                  |          | 1 = Timer callback is enabled                                               |

### For picture controls, rings, and buttons

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_FIT_MODE</td>
<td>int</td>
<td>VAL_SIZE_TO_IMAGE or VAL_SIZE_TO_PICTURE or VAL_PICT_CORNER or VAL_PICT_CENTER or VAL_PICT_TILE</td>
</tr>
</tbody>
</table>

### For picture controls and rings

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ATTR_FRAME_VISIBLE | int      | 1 = show picture frame  
|                  |          | 0 = hide picture frame    |

### For picture controls, rings, and canvas controls

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_PICT_BGCOLOR</td>
<td>int</td>
<td>RGB value—see discussion that follows this table. Also applies to canvas controls</td>
</tr>
</tbody>
</table>
### For picture buttons (command and toggle)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_IMAGE_FILE</td>
<td>char *</td>
<td>The filename of the image file to load into control. A NULL results in no image.</td>
</tr>
<tr>
<td>ATTR_IMAGE_FILE_LENGTH</td>
<td>int</td>
<td>The length of the image file length.</td>
</tr>
<tr>
<td>ATTR_SUBIMAGE_HEIGHT</td>
<td>int</td>
<td>Pixel height of subimage. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_SUBIMAGE_TOP</td>
<td>int</td>
<td>Top pixel coordinate of subimage. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_SUBIMAGE_LEFT</td>
<td>int</td>
<td>Left pixel coordinate of subimage. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_SUBIMAGE_WIDTH</td>
<td>int</td>
<td>Pixel width of subimage. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_USE_SUBIMAGE</td>
<td>int</td>
<td>1 = enable subimage display. 0 = disable subimage display. See discussion following this table.</td>
</tr>
</tbody>
</table>

#### Picture Button Subimage Discussion

ATTR_USE_SUBIMAGE allows the picture button to display a subset of its loaded image. The subset of the image that is sized to the picture button is specified by the following attributes.

ATTR_SUBIMAGE_TOP
ATTR_SUBIMAGE_LEFT
ATTR_SUBIMAGE_WIDTH
ATTR_SUBIMAGE_HEIGHT

ATTR_SUBIMAGE_TOP and ATTR_SUBIMAGE_LEFT define the horizontal and vertical offsets in pixels of the picture button's bitmap image that will be displayed relative to the origin of the bitmap. The bitmap subimage origin (0,0) is the top-left corner of the bitmap.

ATTR_SUBIMAGE_WIDTH and ATTR_SUBIMAGE_HEIGHT define the width and height in pixels for the displayed subimage.

ATTR_USE_SUBIMAGE does not work with images loaded from Windows metafiles (.WMF).
Control Attribute Discussion

An RGB value is a 4-byte integer with the hexadecimal format \(0x00RRGGBB\). \(RR\), \(GG\), and \(BB\) are the respective red, green, and blue components of the color value. You can use the User Interface Library function, `MakeColor`, to create an RGB value from red, green, and blue color components. See Table 3-3 for a list of common color values.

Table 3-10 contains the control types that can be used with the `ATTR_CTRL_STYLE` attribute.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerics</td>
<td><code>CTRL_NUMERIC</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_THERMOMETER</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_TANK</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_GAUGE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_METER</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_KNOB</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_DIAL</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_VSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_HSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_FLAT_VSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_FLAT_HSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_LEVEL_VSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_LEVEL_HSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_POINTER_VSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_NUMERIC_POINTER_HSLIDE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>CTRL_COLOR_NUMERIC</code></td>
<td></td>
</tr>
</tbody>
</table>

(continues)
Table 3-10. Control Styles for ATTR_CTRL_STYLE (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>CTRL_STRING</td>
<td></td>
</tr>
<tr>
<td>Text message</td>
<td>CTRL_TEXT_MSG</td>
<td></td>
</tr>
<tr>
<td>Text box</td>
<td>CTRL_TEXT_BOX</td>
<td></td>
</tr>
<tr>
<td>Command buttons</td>
<td>CTRL_SQUARE_COMMAND_BUTTON</td>
<td><img src="CTRL_SQUARE_COMMAND_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_OBLONG_COMMAND_BUTTON</td>
<td><img src="CTRL_OBLONG_COMMAND_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUND_COMMAND_BUTTON</td>
<td><img src="CTRL_ROUND_COMMAND_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_PICTURE_COMMAND_BUTTON</td>
<td><img src="CTRL_PICTURE_COMMAND_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUNDED_COMMAND_BUTTON</td>
<td><img src="CTRL_ROUNDED_COMMAND_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td>Buttons</td>
<td>CTRL_ROUND_BUTTON</td>
<td><img src="CTRL_ROUND_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_SQUARE_BUTTON</td>
<td><img src="CTRL_SQUARE_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUND_FLAT_BUTTON</td>
<td><img src="CTRL_ROUND_FLAT_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_SQUARE_FLAT_BUTTON</td>
<td><img src="CTRL_SQUARE_FLAT_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUND_RADIO_BUTTON</td>
<td><img src="CTRL_ROUND_RADIO_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_SQUARE_RADIO_BUTTON</td>
<td><img src="CTRL_SQUARE_RADIO_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_CHECK_BOX</td>
<td><img src="CTRL_CHECK_BOX" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUND_PUSH_BUTTON</td>
<td><img src="CTRL_ROUND_PUSH_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_SQUARE_PUSH_BUTTON</td>
<td><img src="CTRL_SQUARE_PUSH_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUND_PUSH_BUTTON2</td>
<td><img src="CTRL_ROUND_PUSH_BUTTON2" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_SQUARE_PUSH_BUTTON2</td>
<td><img src="CTRL_SQUARE_PUSH_BUTTON2" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_SQUARE_TEXT_BUTTON</td>
<td><img src="CTRL_SQUARE_TEXT_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_OBLONG_TEXT_BUTTON</td>
<td><img src="CTRL_OBLONG_TEXT_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUND_TEXT_BUTTON</td>
<td><img src="CTRL_ROUND_TEXT_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_ROUNDED_TEXT_BUTTON</td>
<td><img src="CTRL_ROUNDED_TEXT_BUTTON" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td>CTRL_PICTURE_TOGGLE_BUTTON</td>
<td><img src="CTRL_PICTURE_TOGGLE_BUTTON" alt="Image" /></td>
</tr>
</tbody>
</table>

(continues)
Table 3-10. Control Styles for ATTR_CTRL_STYLE (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LEDs</strong></td>
<td><strong>CTRL_ROUND_LIGHT</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_SQUARE_LIGHT</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_ROUND_LED</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_SQUARE_LED</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Binary</strong></td>
<td><strong>switches</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_HSWITCH</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_VSWITCH</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_GROOVED_HSWITCH</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_GROOVED_VSWITCH</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_TOGGLE_HSWITCH</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_TOGGLE_VSWITCH</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Rings</strong></td>
<td><strong>CTRL_RING</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RECESSED_MENU_RING</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_MENU_RING</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_POPUP_MENU_RING</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_VSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_HSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_FLAT_VSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_FLAT_HSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_LEVEL_VSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_LEVEL_HSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_POINTER_VSLIDE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CTRL_RING_POINTER_HSLIDE</strong></td>
<td></td>
</tr>
</tbody>
</table>

(continues)
Table 3-10. Control Styles for ATTR_CTRL_STYLE (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CTRL_RING_THERMOMETER</td>
<td>![thermometer]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RING_TANK</td>
<td>![tank]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RING_GAUGE</td>
<td>![gauge]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RING_METER</td>
<td>![meter]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RING_KNOB</td>
<td>![knob]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RING_DIAL</td>
<td>![dial]</td>
</tr>
<tr>
<td></td>
<td>CTRL_PICTURE_RING</td>
<td>![picture]</td>
</tr>
<tr>
<td>List box</td>
<td>CTRL_LIST</td>
<td></td>
</tr>
<tr>
<td>Decorations</td>
<td>CTRL_RAISED_BOX</td>
<td>![raised_box]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RECESSED_BOX</td>
<td>![recursed_box]</td>
</tr>
<tr>
<td></td>
<td>CTRL_FLAT_BOX</td>
<td>![flat_box]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RAISED_CIRCLE</td>
<td>![raised_circle]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RECESSED_CIRCLE</td>
<td>![recursed_circle]</td>
</tr>
<tr>
<td></td>
<td>CTRL_FLAT_CIRCLE</td>
<td>![flat_circle]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RAISED_FRAME</td>
<td>![raised_frame]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RECESSED_FRAME</td>
<td>![recursed_frame]</td>
</tr>
<tr>
<td></td>
<td>CTRL_FLAT_FRAME</td>
<td>![flat_frame]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RAISED_ROUNDED_FRAME</td>
<td>![raised_rounded_frame]</td>
</tr>
<tr>
<td></td>
<td>CTRL_RECESSED_ROUNDED_FRAME</td>
<td>![recursed_rounded_frame]</td>
</tr>
<tr>
<td></td>
<td>CTRL_FLAT_ROUNDED_BOX</td>
<td>![flat_rounded_box]</td>
</tr>
</tbody>
</table>

(continues)
Table 3-10. Control Styles for ATTR_CTRL_STYLE (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>CTRL_GRAPH</td>
<td></td>
</tr>
<tr>
<td>Strip chart</td>
<td>CTRL_STRIP_CHART</td>
<td></td>
</tr>
<tr>
<td>Picture</td>
<td>CTRL_PICTURE</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td>CTRL_TIMER</td>
<td></td>
</tr>
<tr>
<td>Canvas</td>
<td>CTRL_CANVAS</td>
<td></td>
</tr>
</tbody>
</table>

The following list presents the control data types that correspond to the ATTR_DATA_TYPE attribute.

Table 3-11. Control Data Types for the ATTR_DATA_TYPE attribute

<table>
<thead>
<tr>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_CHAR</td>
</tr>
<tr>
<td>VAL_INTEGER</td>
</tr>
<tr>
<td>VAL_SHORT_INTEGER</td>
</tr>
<tr>
<td>VAL_FLOAT</td>
</tr>
<tr>
<td>VAL_DOUBLE</td>
</tr>
<tr>
<td>VAL_STRING</td>
</tr>
<tr>
<td>VAL_UNSIGNED_SHORT_INTEGER</td>
</tr>
<tr>
<td>VAL_UNSIGNED_INTEGER</td>
</tr>
<tr>
<td>VAL_UNSIGNED_CHAR</td>
</tr>
</tbody>
</table>

You should set ATTR_DATA_TYPE before you set any other value attribute such as ATTR_CTRL_VAL, ATTR_MAX_VALUE, ATTR_MIN_VALUE, and others.
Table 3-12 contains the numeric formats you can use with ATTR_FORMAT.

Table 3-12. Numeric Formats

<table>
<thead>
<tr>
<th>Numeric Formats</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_FLOATING_PT_FORMAT</td>
<td>123.000</td>
</tr>
<tr>
<td>VAL_SCIENTIFIC_FORMAT</td>
<td>1.23E+2</td>
</tr>
<tr>
<td>VAL_ENGINEERING_FORMAT</td>
<td>123.00E+0</td>
</tr>
<tr>
<td>VAL_DECIMAL_FORMAT*</td>
<td>123</td>
</tr>
<tr>
<td>VAL_HEX_FORMAT*</td>
<td>7B</td>
</tr>
<tr>
<td>VAL_OCTAL_FORMAT*</td>
<td>173</td>
</tr>
<tr>
<td>VAL_BINARY_FORMAT*</td>
<td>1111011</td>
</tr>
</tbody>
</table>

* not valid for graphs and strip charts

The ATTR_CHECK_RANGE attribute establishes the behavior of LabWindows/CVI when you try to set a control to a value outside of its specified range. The three possible attribute values are shown in the following table.

Table 3-13. ATTR_CHECK_RANGE Values

<table>
<thead>
<tr>
<th>Numeric Formats</th>
<th>LabWindows/CVI Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_COERCE</td>
<td>The value is coerced to the upper or lower range boundary, whichever is closer.</td>
</tr>
<tr>
<td>VAL_IGNORE</td>
<td>The value remains unchanged.</td>
</tr>
<tr>
<td>VAL_NOTIFY</td>
<td>The user interface operator is notified with an Out of Range dialog box when the control is active.</td>
</tr>
</tbody>
</table>

Programming with Picture Controls

This section describes the simple picture control and the picture control versions of command buttons, toggle buttons, and ring controls onto which you can place images. Picture controls can help users understand your GUI more quickly and can be used to display useful diagrams or
charts. Picture controls can include logos and other images that match the style of your organization.

You can use commit events on any picture control in your program. However, the simple picture control is best suited for displaying images only. The other picture controls have a powerful editing window that helps you quickly set their appearance and their behavior in response to user events.

The following image formats work with picture controls:

- PCX (Windows and UNIX)
- BMP, DIB, RLE, ICO (Windows only)
- WMF (Windows 95 and NT only)
- XWD (UNIX only)

**Simple Picture Control**

You access the simple picture control through the **Picture** menu item in the **Create** menu. Alternatively, you can right-click on a panel in a .uir file and select **Picture** from the pop-up menu that appears. It allows you to place images on panels, such as logos and diagrams.

You can program the simple picture control to recognize user events, but it does not automatically change appearance in response to the event. The simple picture control can also serve as an indicator that, for example, is dim at startup and becomes fully colored at some point during run-time. You can programatically change the contents of a picture control by calling **SetCtrlAttribute** with the **ATTR_IMAGE_FILE** attribute.

**Picture Control Attributes**

You access the picture command button, picture toggle button, and picture ring control under the **Control** menu, within the sub-menus **Command Button**, **Toggle Button**, and **Ring**. The picture versions of these controls have nearly all the features of other controls.

You can use the **SetCtrlVal** function to change the image in a picture button or picture ring, when, for example, you want to simulate a mouse click. For picture ring controls, you also can use the **SetCtrlIndex** function to change the image. You can use the **SetCtrlBitmap** function to select images directly from memory, instead of through a path to a standard image filename. **SetCtrlBitmap** is an alternative to **ATTR_IMAGE_FILE**, **DisplayImageFile** and **ReplaceListItem**. For more information, see the **Using Bitmap Objects** section in this chapter.
Appearance of Picture Controls

The picture controls appear on screen like other controls, with the following minor differences.

- Picture command buttons and picture toggle buttons have an external label only.

- In picture command buttons and picture toggle buttons, the background color of your buttons may not be visible, depending on the size of your image and the Fit Mode you choose. When the background color of a button is not visible, users do not see the color shift that normally takes place in response to a mouse click. However, they do see two smaller changes in the button: the image moves down and over several pixels, and the colors of the border around the button change. You can give these controls more visual impact by making the background of the control visible and coloring it appropriately.

- In a picture ring control, you cannot click on one of the images and view all the selections at once. However, as with other ring controls, you can cycle through the options in the ring control by using increment/decrement arrows.

Giving Picture Controls More Visual Impact

Whether the state of the control is on, off, or “clicked-on with the mouse,” users see the same image. If you want your picture control to reflect its state more vividly, make the background area of the control visible by choosing the Fit Modes Stick Image to Corner or Center Image when you paste the image into the control. As long as your image is smaller than the size of your control, the image has a visible background area that you can color vividly to call attention to the state of a picture control. Either resize the control or resize the image until enough background appears on your control.

Programming with Canvas Controls

Use a canvas control to add an arbitrary drawing surface to your project. You can draw text, shapes, and bitmap images. This section describes how you can use the User Interface Library functions and attributes with canvas controls.

Functions for Drawing on Canvas

Use the following functions to draw on a canvas.

- CanvasDrawPoint to draw a point.
- CanvasDrawLine to draw a line.
- CanvasDrawLineTo to draw a line from the current pen position.
- CanvasDrawRect to draw a rectangle.
• CanvasDimRect to overlay a checkerboard pattern in a rectangular area.
• CanvasDrawRoundedRect to draw a rectangle with rounded corners.
• CanvasDrawOval to draw an oval.
• CanvasDrawArc to draw an arc.
• CanvasDrawPoly to draw a polygon.
• CanvasDrawText to draw text within a rectangular area.
• CanvasDrawTextAtAPoint to draw text at an anchor point.
• CanvasDrawBitmap to draw a bitmap image.
• CanvasScroll to scroll a rectangular area.
• CanvasInvertRect to invert the colors in a rectangular area.
• CanvasClear to restore a rectangular area to the canvas background color.

Batch Drawing

Although, the drawing functions can be called at any time, they are most efficient when called from within a batch drawing operation. A batch drawing operation consists of a call to CanvasStartBatchDraw, followed by one or more calls to the canvas drawing functions, followed by a call to CanvasEndBatchDraw.

For optimal performance, users are encouraged to include as many drawing primitives as possible within a batch drawing operation. When a drawing function is called outside of a batch operation, the function is implicitly surrounded by calls to CanvasStartBatchDraw and CanvasEndBatchDraw.

Canvas Coordinate System

A canvas has a built-in pixel-based Cartesian coordinate system, where (0,0) represents the top, left corner of the canvas. All drawing is specified relative to this coordinate system. The coordinate system can be modified using the following four attributes:

ATTR_CANVAS_XCOORD_AT_ORIGIN
ATTR_CANVAS_YCOORD_AT_ORIGIN
ATTR_CANVAS_XSCALING
ATTR_CANVAS_YSCALING

All canvas control functions use this coordinate system, except for CanvasGetPixel and CanvasGetPixels, which use unscaled pixel coordinates rather than the canvas coordinate system.
Offscreen Bitmap

Each canvas has an offscreen bitmap which is used to restore the appearance of the canvas when the region is exposed. You can choose to draw directly to the screen, bypassing the offscreen bitmap. If you draw to the offscreen bitmap, you can choose whether to update the screen immediately or wait until draw events are processed. This is controlled by the ATTR_DRAW_POLICY attribute.

The CanvasUpdate function immediately copies the canvas offscreen bitmap to the screen, within a specified rectangular area.

The ATTR_OVERLAP_POLICY attribute controls what occurs when you draw to a canvas which is overlapped by another control.

Clipping

The drawing functions are constrained by the clipping set using CanvasSetClipRect. Any drawing outside the clipping rectangle is not rendered. You can obtain the current clipping rectangle by calling CanvasSetClipRect.

Background Color

The background color of the canvas is controlled by the ATTR_PICT_BGCOLOR attribute. When ATTR_PICT_BGCOLOR is changed, the entire canvas area is cleared.

Pens

Each canvas has a pen. The canvas pen attributes can be set individually using SetCtrlAttribute. They are:

ATTR_PEN_WIDTH
ATTR_PEN_STYLE
ATTR_PEN_COLOR
ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN

The CanvasDefaultPen function resets all these attributes to their default values.

The location of the pen affects the starting position of the line drawn by the CanvasDrawLineTo function. The location of the pen is affected only by the CanvasSetPenPosition and the CanvasDrawLineTo functions. You can obtain the location of the pen by calling CanvasGetPenPosition.
Pixel Values

You can obtain the color values of pixels in the canvas. Call `CanvasGetPixel` to obtain the color of one pixel. Call `CanvasGetPixels` to obtain the values of the pixels within a rectangular area. The color values are obtained from the offscreen bitmap, not the screen.

Unlike other canvas control functions, `CanvasGetPixel` and `CanvasGetPixels` use unscaled pixel coordinates rather than the canvas coordinate system.

Canvas Attribute Discussion

The following table lists the attributes for Canvas Controls.

Table 3-14. Control Attributes for Canvas Controls

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_DRAW_POLICY</td>
<td>int</td>
<td>Determines when drawing operations are rendered on the offscreen bitmap and the screen. See discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_OVERLAPPED_POLICY</td>
<td>int</td>
<td>Determines what occurs when you draw to a canvas which is overlapped by another control. See discussion that follows this table.</td>
</tr>
<tr>
<td>ATTR_PEN_COLOR</td>
<td>int</td>
<td>The RGB color value used to draw points, lines, frames, and text on the canvas.</td>
</tr>
<tr>
<td>ATTR_PEN_FILL_COLOR</td>
<td>int</td>
<td>The RGB color value used to fill interior areas of shapes, text backgrounds, and areas exposed by scrolling.</td>
</tr>
<tr>
<td>ATTR_PEN_MODE</td>
<td>int</td>
<td>Determines the effect of drawing with the pen color (or pen fill color), given the current color on the screen. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_PEN_PATTERN</td>
<td>unsigned char[8]</td>
<td>Determines the pattern used to fill interior areas of shapes. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_PEN_STYLE</td>
<td>int</td>
<td>The style used when drawing lines and frames. In Windows, applies only if pen width is 1; if pen width is greater than 1, the style is always VAL_SOLID. See Table 3-23.</td>
</tr>
</tbody>
</table>

(continues)
Table 3-14. Control Attributes for Canvas Controls (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_PEN_WIDTH</td>
<td>int</td>
<td>The number of pixels in the width of a pen stroke. Applies to lines, frames, and points. Valid range: 1 to 255. Default: 1.</td>
</tr>
<tr>
<td>ATTR_XCOORD_AT_ORIGIN</td>
<td>double</td>
<td>The horizontal coordinate mapped to the left edge of the canvas. (Is multiplied by ATTR_XSCALING to arrive at a pixel offset.) Default value: 0.0.</td>
</tr>
<tr>
<td>ATTR_XSCALING</td>
<td>double</td>
<td>The factor used to scale user-supplied horizontal coordinates and widths into pixel-based coordinates and widths. Default value: 1.0.</td>
</tr>
<tr>
<td>ATTR_YCOORD_AT_ORIGIN</td>
<td>double</td>
<td>The vertical coordinate mapped to the top edge of the canvas. (Is multiplied by ATTR_YSCALING to arrive at a pixel offset.) Default value: 0.0.</td>
</tr>
<tr>
<td>ATTR_YSCALING</td>
<td>double</td>
<td>The factor used to scale user-supplied vertical coordinates and heights into pixel-based coordinates and heights. Default value: 1.0.</td>
</tr>
</tbody>
</table>

The following table lists the values associated with the ATTR_DRAW_POLICY attribute.

Table 3-15. Values for ATTR_DRAW_POLICY

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_UPDATE_IMMEDIATELY</td>
<td>Drawing takes place offscreen. The section of the bitmap corresponding to the area of the drawing operation is copied into the canvas display immediately. (This is the default.)</td>
</tr>
<tr>
<td>VAL_MARK_FOR_UPDATE</td>
<td>Drawing takes place offscreen. The area on the canvas corresponding to the area of the drawing operation is marked for update. The new drawing becomes visible when draw events are processed.</td>
</tr>
<tr>
<td>VAL_DIRECT_TO_SCREEN</td>
<td>Drawing goes directly to the screen. The offscreen bitmap is not updated. Although this may result in a faster drawing time, whatever is drawn in this mode is lost when the canvas is redrawn.</td>
</tr>
</tbody>
</table>
The following table lists the values associated with the `ATTR_OVERLAPPED_POLICY` attribute.

**Table 3-16. Values for ATTR_OVERLAPPED_POLICY**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_DEFER_DRAWING</td>
<td>If the control is overlapped and the draw policy is VAL_UPDATE_IMMEDIATELY, new drawing does not become visible until draw events are processed. If the draw policy is VAL_DIRECT_TO_SCREEN, no drawing takes place at all. (This is the default.)</td>
</tr>
<tr>
<td>VAL_DRAW_ON_TOP</td>
<td>If the control is overlapped and the draw policy is <em>not</em> VAL_MARK_FOR_UPDATE, drawing occurs on top of the overlapping controls.</td>
</tr>
</tbody>
</table>

The `ATTR_PEN_MODE` attribute determines the effect of drawing with the pen color (or pen fill color), given the current color on the screen. With the default setting, VAL_COPY_MODE, the current screen color is replaced with the pen color (or pen fill color). The other settings specify bitwise logical operations on pen color (or pen fill color) and the screen color.

**Note:** *If a system color palette is in use, the logical operations might be performed on the palette indices rather than the RGB values, depending on the operating system.*

The following table lists the values associated with the `ATTR_PEN_MODE` attribute.

**Table 3-17. Values for ATTR_PEN_MODE**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_COPY_MODE</td>
<td>pen color (the default)</td>
</tr>
<tr>
<td>VAL_OR_MODE</td>
<td>pen color</td>
</tr>
<tr>
<td>VAL_XOR_MODE</td>
<td>pen color ^ screen color</td>
</tr>
<tr>
<td>VAL_AND_NOT_MODE</td>
<td>~(pen color) &amp; screen color</td>
</tr>
<tr>
<td>VAL_NOT_COPY_MODE</td>
<td>~(pen color)</td>
</tr>
<tr>
<td>VAL_OR_NOT_MODE</td>
<td>~(pen color)</td>
</tr>
<tr>
<td>VAL_NOT_XOR_MODE</td>
<td>~(pen color ^ screen color)</td>
</tr>
<tr>
<td>VAL_AND_MODE</td>
<td>pen color &amp; screen color</td>
</tr>
</tbody>
</table>

The `ATTR_PEN_PATTERN` attribute determines the pattern used to fill interior areas of shapes. The value is an 8-byte unsigned character array representing a repeating 8-by-8 grid of pixels through which filling operations are filtered. A pixel of value 1 means that the pen fill color is
used for that pixel. A pixel value of 0 means that black is used for that pixel. The default value for the attribute is the solid pattern, in which each byte of the array is 0xFF.

To make a pixel value of 0 mean "screen color" instead of "black", do the following.

1. Set ATTR_PEN_PATTERN to the complement of the pattern you wish to use.

2. Set ATTR_PEN_MODE to VAL_AND_MODE.

3. Set ATTR_PEN_FILL_COLOR to VAL_WHITE.

4. Use a canvas draw function, (for example, CanvasDrawRect) to fill the area.

5. Set ATTR_PEN_PATTERN to the desired pattern.

6. Change the ATTR_PEN_MODE to VAL_OR_MODE.

7. Change the ATTR_PEN_FILL_COLOR to the desired pattern color.

8. Draw again.

**Using Rect and Point Structures**

Two structures, Rect and Point are defined in the userint.h include file. These structures are used to specify locations and areas in Cartesian coordinate systems, such as those used in canvas controls and bitmaps. Many canvas control functions use these structures.

The Rect structure specifies the location and size of a rectangle. It is defined as follows.

```c
typedef struct
{
    int top;
    int left;
    int height;
    int width;
} Rect;
```

A Point structure specifies the location of a point. It is defined as follows.

```c
typedef struct
{
    int x;
    int y;
} Point;
```
Functions and Macros for Making Rects and Points

You might need to create a `Rect` or `Point` just to pass it to a function. You can avoid creating a variable for this by using one of the following functions.

```c
Rect MakeRect (int top, int left, int height, int width);
Point MakePoint (int x, int y);
```

For example,

```c
CanvasDrawPoint (panel, ctrl, MakePoint (30, 40));
```

You can also use these function to initialize variables. For example,

```c
Rect r = MakeRect (10, 20, 100, 130);
```

There are special values for the `Rect` height and width. Also, there are some macros for creating commonly used rectangles. The documentation for each function indicates when these values and macros are applicable. See the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value or Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_TO_EDGE</td>
<td>-1</td>
<td>Set the <code>Rect</code> width (or height) to the distance from the <code>Rect</code> left (or top) to the right (or bottom) edge of the object.</td>
</tr>
<tr>
<td>VAL_KEEP_SAME_SIZE</td>
<td>-2</td>
<td>When copying objects (such as bitmaps), make the destination object the same size as the source object.</td>
</tr>
<tr>
<td>VAL_EMPTY_RECT</td>
<td>MakeRect (0, 0, 0, 0)</td>
<td>An empty rectangle.</td>
</tr>
<tr>
<td>VAL_ENTIRE_OBJECT</td>
<td>MakeRect (0, 0, VAL_TO_EDGE, VAL_TO_EDGE)</td>
<td>Make the <code>Rect</code> the size of the object (for example, the canvas or bitmap).</td>
</tr>
</tbody>
</table>

Functions for Modifying Rects and Points

Use the following functions to set or modify the values in a `Rect` or `Point` structure.

- `RectSet` to set each of the four values of an existing `Rect` structure.
- `RectSetFromPoints` to set a `Rect` so that it defines the smallest rectangle that encloses two points.
- `RectSetBottom` to set the height of a `Rect` so that the bottom is a given value. (The bottom is not enclosed by the rectangle.)
• **RectSetRight** to set the height of a Rect so that the right edge is a given value. (The right edge is *not* enclosed by the rectangle.)

• **RectSetCenter** to set the top and left of a Rect so that it is centered around a given value, while keeping the same size.

• **RectOffset** to modify the top and left of a Rect so as to shift the location of the rectangle.

• **RectMove** to set the top and left of a Rect to a given Point.

• **RectGrow** to modify the values in a Rect so that the rectangle grows or shrinks around its current center point.

• **PointSet** to set the two values in an existing Point structure.

### Functions for Comparing or Obtaining Values from Rects and Points

Use the following functions to compare or obtain values from a Rect or Point structure.

• **RectBottom** to obtain the location of the bottom of a rectangle.

• **RectRight** to obtain the location of the right edge of a rectangle.

• **RectCenter** to obtain the location of the center of a rectangle.

• **RectEqual** to determine if two rectangles are identical.

• **RectEmpty** to determine if a rectangle is empty.

• **RectContainsPoint** to determine if a rectangle encloses a given point.

• **RectContainsRect** to determine if a rectangle completely encloses another rectangle.

• **RectSameSize** to determine if two rectangles are the same size.

• **RectUnion** to set a Rect to the smallest rectangle that encloses two given rectangles.

• **RectIntersection** to set a Rect to the largest rectangle that is enclosed by two given rectangles.

• **PointEqual** to determine if two points are at the same location.

• **PointPinnedToRect** to modify a Point structure, if needed, to ensure that it is within a given rectangle.

### Using Bitmap Objects

A bitmap is a two-dimensional grid of pixels representing an image. There are some functions, such as **PlotBitmap** (for graph controls) and **DisplayImageFile** (for picture controls) which read an image out of a file and directly display it on a control.
There are other functions, however, which create or extract a bitmap, store them in memory, and return a bitmap ID. You can then use the bitmap ID in other functions.

**Functions for Creating, Extracting, or Discarding Bitmap Objects**

Use the following functions to create, extract, or discard bitmap objects.

- **NewBitmap** to create a bitmap object from scratch.
- **GetBitmapFromFile** to create a bitmap object using image data read from a file.
- **GetCtrlBitmap** to create a bitmap object from an image contained in a picture, picture ring, picture button, canvas, or graph control.
- **GetCtrlDisplayBitmap** to create a bitmap object from the current appearance of a control.
- **GetPanelDisplayBitmap** to create a bitmap object from the current appearance of a specified rectangular area of a panel.
- **ClipboardGetBitmap** to create a bitmap object from an image (if any) in the system clipboard.
- **DiscardBitmap** to remove a bitmap from memory.

If you want to display images that are not rectangular or that have “holes” in them, you can use bitmaps that have a transparent background. If you are creating your bitmap image from scratch, you can achieve transparency by using the **mask** parameter to the **NewBitmap** function.
Windows Metafiles

A Windows metafile (.WMF) contains a description of an image that is scaleable without distortion. The description consists of a set of drawing commands rather than a bitmap. Windows metafiles are available only on Windows 95 and NT. You can load them using the GetBitmapFromFile function.

Functions for Displaying or Copying Bitmap Objects

Use the following functions to display a bitmap object in a control or copy an image from a bitmap object to a control.

- **CanvasDrawBitmap** to display a bitmap in a canvas control.
- **SetCtrlBitmap** to set an image in a picture, picture ring, picture button, or graph control from a bitmap object. Can be used to replace an existing image create a new image.
- **ClipboardPutBitmap** to copy image data from a bitmap object to the system clipboard.

Functions for Retrieving Image Data from Bitmap Objects

Use the following functions to retrieve image data from bitmap objects.

- **GetBitmapInfo** to obtain size information about the image associated with a bitmap. This information can then be used in allocating the buffers to be passed to **GetBitmapData**.
- **AllocBitmapData** to allocate the buffers necessary for calling **GetBitmapData**. This is an alternative to calling **GetBitmapInfo** and allocating the buffers yourself.
- **GetBitmapData** to obtain the bit values that define the image associated with a bitmap.

Programming with Timer Controls

This section describes the functions and attributes that you can use to manipulate timer control activity.

Timer Control Functions

You can create timer controls with the User Interface Editor or by using the function **NewCtrl**. There is no limit to the number of timer controls you can use, but timer callbacks are called sequentially. If two or more timers have identical time intervals, their callbacks are called in an undefined, but consistent order.
Use `SuspendTimerCallbacks` to stop calls to all the timer control callbacks until `ResumeTimerCallbacks` restores timer control activity. These functions do not affect the interval schedules or set the enabled/disabled state of individual timer controls, but rather activate and deactivate a blanket suspension over all timer callbacks.

Use `ResetTimer` to reset the interval start times of individual timer controls, all timer controls on a panel, or all timer controls on all panels.

### Using Timer Callbacks

The timer callback has the same prototype as other control callbacks.

```c
int CVICALLBACK timerfunc (int panel, int control, int event, 
                            void *callbackData, int eventData1, 
                            int eventData2)
```

`eventData1` is a pointer to a `double` that represents the current time in seconds. The time base is the same as that used by the `Timer` function in the Utility Library. `eventData2` is a pointer to a `double` that represents the time that has elapsed since the last call to the timer callback. The elapsed time is set to zero if the callback has not been called previously.

When the callback function is called at the end of an interval, `event` is `EVENT_TIMER_TICK`.

### Timer Control Attributes

Table 3-19 lists the timer control attributes you can retrieve or change using `SetCtrlAttribute` and `GetCtrlAttribute`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_INTERVAL</td>
<td>double</td>
<td>The time interval of the timer control in seconds.</td>
</tr>
</tbody>
</table>
| ATTR_ENABLED    | int  | 0 = timer control is disabled  
                         1 = timer control is enabled |
Timer Control Attribute Discussion

ATTR_INTERVAL sets the time interval of the timer control in seconds. An interval of zero results in timer events occurring as fast as CVI can generate them. Setting the interval less than the system clock resolution results in an interval equivalent to the system clock resolution. If the timer has already been started, setting ATTR_INTERVAL resets the timer. The ATTR_INTERVAL default value is one second.

Note: The time intervals you specify are minimum values. System activity, including processing of user callbacks, can cause timer callbacks to be late or skipped. If you have a need for real time response, do not write into your program operations that take significant amounts of time without giving CVI a chance to process events.

ATTR_ENABLED determines whether a timer control’s callback is called at each interval. The ATTR_ENABLED default value is TRUE.

Details of Timer Control Operations

After a timer control is created or loaded, the timer does not start until a call is made to RunUserInterface, GetUserEvent, or ProcessSystemEvents. This ensures that you can create or load several timer controls and have them start at the same time.

The timer interval schedules are not affected by SuspendTimerCallbacks, ResumeTimerCallbacks, or by changing the value of ATTR_ENABLED. You can reset timer interval schedules by calling ResetTimer. The length of the timer intervals can be changed using the ATTR_INTERVAL attribute. Changing the ATTR_INTERVAL value also causes the interval schedule for the timer control to be reset.

Timer callbacks are not called unless a call to RunUserInterface, GetUserEvent, or ProcessSystemEvents is in effect. Calling RunUserInterface, GetUserEvent, and ProcessSystemEvents does not alter interval schedules already in effect, but may trigger an overdue callback. If, upon calling one of these functions, the time since the last callback is greater than the ATTR_INTERVAL value for a timer control, the callback is called. Only one such overdue callback is called no matter how many intervals have elapsed. Overdue callbacks do not cause the next interval to be rescheduled. Thus, the time between the overdue callback and the next regularly scheduled callback may be less than the ATTR_INTERVAL value.

Calling ResumeTimerCallbacks or setting ATTR_ENABLED to TRUE does not trigger overdue callbacks.

Programming with Graph and Strip Chart Controls

This section describes how you can use the User Interface Library functions to control the elements of user interface graphs and strip charts.
Functions for Graphs and Strip Charts

As with any other control, you can create graphs and strip charts in the User Interface Editor or programmatically using NewCtrl. For details, see the Programming with Controls section of this chapter. You also use GetCtrlAttribute and SetCtrlAttribute to control the attributes of graphs and strip charts. The following functions are provided to programmatically change the axis ranges of graphs and strip charts and to change the scaling mode.

Use GetAxisRange to obtain the current x and y axis ranges of a graph or strip chart control.

Use SetAxisRange to set the current x and y axis ranges of a graph or strip chart control.

Functions for Graphs Only

Use one of the following functions to add a plot to a graph.

- Use PlotArc to plot an arc.
- Use PlotBitmap to plot a bitmap image.
- Use PlotIntensity or PlotScaledIntensity to plot a color-intensity graph.
- Use PlotLine to plot a single line segment.
- Use PlotOval to plot an oval.
- Use PlotPoint to plot a point.
- Use PlotPolygon to plot a polygon.
- Use PlotRectangle to plot a rectangle.
- Use PlotText to plot a text string.
- Use PlotWaveform to plot a waveform array.
- Use PlotX to plot X versus its indices.
- Use PlotXY to plot X versus Y.
- Use PlotY to plot Y versus its indices.

If the graph is visible, the program draws the plot immediately.

Use DeleteGraphPlot to remove one or all plots from a graph. DeleteGraphPlot frees the appropriate plots from memory.
Use `GetPlotAttribute` to obtain an attribute of a particular graph plot.

Use `SetPlotAttribute` to set an attribute of a particular graph plot.

Use one of the following functions to control the cursors in a graph.

- Use `GetGraphCursor` to obtain the current position of a specified cursor.
- Use `SetGraphCursor` to set the position of a specified cursor.
- Use `GetGraphCursorIndex` to obtain the plot handle and the array index of the plot that the specified cursor is attached to.
- Use `SetGraphCursorIndex` to attach a cursor to a particular data point.
- Use `GetActiveGraphCursor` to obtain the index of the active graph cursor.
- Use `SetActiveGraphCursor` to set the active graph cursor.
- Use `GetCursorAttribute` to obtain an attribute of a specified cursor.
- Use `SetCursorAttribute` to an attribute of a specified cursor.

**Functions for Strip Charts Only**

- Use `PlotStripChart` to add one or more points to the strip chart traces. If the strip chart is visible, the points are drawn as you add them to each trace.
- Use `PlotStripChartPoint` to add one point to a strip chart that contains exactly one trace.
- Use `ClearStripChart` to clear all points from the strip chart. If the strip chart is visible, the points on the plot clear.
- Use `GetTraceAttribute` to obtain an attribute of a particular strip chart trace
- Use `SetTraceAttribute` to set an attribute of a particular strip chart trace.

**Processing Graph and Strip Chart Events**

Graph events are processed like any other control. See the Processing Control Events section of this chapter for details of graph event processing. When the user moves a cursor on a graph a commit event is generated.
Because strip charts do not generate commit events, you can only process events from a strip chart through a callback function. See the *Processing Control Events* section of this chapter for details about processing events using callbacks.

### Graph and Strip Chart Attributes

Table 3-20 lists control attributes you can access through `GetCtrlAttribute` and `SetCtrlAttribute`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_BORDER_VISIBLE</td>
<td>int</td>
<td>1 = border visible 0 = border invisible</td>
</tr>
<tr>
<td>ATTR_EDGE_STYLE</td>
<td>int</td>
<td>VAL_RAISED_EDGE or VAL_FLAT_EDGE or VAL_RECESSED_EDGE</td>
</tr>
<tr>
<td>ATTR_GRAPH_BGCOLOR</td>
<td>int</td>
<td>Graph border background color (RGB value—see discussion that follows this table).</td>
</tr>
<tr>
<td>ATTR_GRID_COLOR</td>
<td>int</td>
<td>Grid color (RGB value—see discussion that follows this table).</td>
</tr>
<tr>
<td>ATTR_INNER_LOG_MARKERS_VISIBLE</td>
<td>int</td>
<td>Specifies whether labels and tick marks are shown next to the inner grid lines of log scale axes. (Default value: FALSE)</td>
</tr>
<tr>
<td>ATTR_PLOT_AREA_HEIGHT</td>
<td>int</td>
<td>The height of the plotting area in pixels (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_PLOT_AREA_WIDTH</td>
<td>int</td>
<td>The width of the plotting area in pixels (GetCtrlAttribute only).</td>
</tr>
<tr>
<td>ATTR_PLOT_BGCOLOR</td>
<td>int</td>
<td>Plot background color (RGB value—see discussion that follows this table).</td>
</tr>
</tbody>
</table>

(continues)
For graphs and strip charts (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_XAXIS_GAIN</td>
<td>double</td>
<td>The factor used to scale the value labels on the X axis. For example, if the X value is 10.0 and ATTR_XAXIS_GAIN is 2.0, then the label on the X axis shows as 20.0. (Default value: 1.0)</td>
</tr>
<tr>
<td>ATTR_XAXIS_OFFSET</td>
<td>double</td>
<td>The amount added to the value labels on the X axis. For example, if the X value is 10.0 and ATTR_XAXIS_OFFSET is 5.0, then the label on the X axis shows as 15.0. The X value is multiplied by ATTR_XAXIS_GAIN before ATTR_XAXIS_OFFSET is added. (Default value: 0.0)</td>
</tr>
<tr>
<td>ATTR_XDIVISIONS</td>
<td>int</td>
<td>(1 to 100, or VAL_AUTO)</td>
</tr>
<tr>
<td>ATTR_XENG_UNITS</td>
<td>int</td>
<td>(-308 to 308)</td>
</tr>
<tr>
<td>ATTR_XFORMAT</td>
<td>int</td>
<td>See Table 3-12.</td>
</tr>
<tr>
<td>ATTR_XGRID_VISIBLE</td>
<td>int</td>
<td>1 = x-grid visible 0 = x-grid invisible</td>
</tr>
<tr>
<td>ATTR_XLABEL_VISIBLE</td>
<td>int</td>
<td>1 = x-label visible 0 = x-label invisible</td>
</tr>
<tr>
<td>ATTR_XNAME</td>
<td>char *</td>
<td>x-axis name.</td>
</tr>
<tr>
<td>ATTR_XUSE_LABEL_STRINGS</td>
<td>int</td>
<td>Whether the X axis numerical value labels are replaced by strings associated with the X values. These strings can be specified either in the User Interface Editor or by calling the InsertAxisItem function.</td>
</tr>
<tr>
<td>ATTR_YAXIS_GAIN</td>
<td>double</td>
<td>The factor used to scale the value labels on the Y axis. For example, if the Y value is 10.0 and ATTR_YAXIS_GAIN is 2.0, then the label on the Y axis shows as 20.0. (Default value: 1.0)</td>
</tr>
</tbody>
</table>

(continues)
### For graphs and strip charts (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_YAXIS_OFFSET</td>
<td>double</td>
<td>The amount added to the value labels on the Y axis. For example, if the Y value is 10.0 and ATTR_YAXIS_OFFSET is 5.0, then the label on the Y axis shows as 15.0. The Y value is multiplied by ATTR_YAXIS_GAIN before ATTR_YAXIS_OFFSET is added. (Default value: 0.0)</td>
</tr>
<tr>
<td>ATTR_YAXIS_REVERSE</td>
<td>int</td>
<td>Whether to reverse the orientation of the Y axis so that the lowest value is shown at the top. If the orientation of the Y axis is reversed, the vertical orientation of the plots is also reversed.</td>
</tr>
<tr>
<td>ATTR_YUSE_LABEL_STRINGS</td>
<td>int</td>
<td>Whether the Y axis numerical value labels are replaced by strings associated with the Y values. These strings can be specified either in the User Interface Editor or by calling the InsertAxisItem function.</td>
</tr>
<tr>
<td>ATTR_XNAME_LENGTH</td>
<td>int</td>
<td>Number of characters in x-axis name; GetCtrlAttribute only.</td>
</tr>
<tr>
<td>ATTR_XPRECISION</td>
<td>int</td>
<td>(0 to 15, or VAL_AUTO)</td>
</tr>
<tr>
<td>ATTR_XYLABEL_BOLD</td>
<td>int</td>
<td>1 = x &amp; y axes labels bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = x &amp; y axes labels not bold</td>
</tr>
<tr>
<td>ATTR_XYLABEL_COLOR</td>
<td>int</td>
<td>x- and y-axis label color (RGB value—see discussion that follows this table).</td>
</tr>
<tr>
<td>ATTR_XYLABEL_FONT</td>
<td>char *</td>
<td>x- and y-axis label font (see Table 3-5).</td>
</tr>
<tr>
<td>ATTR_XYLABEL_FONT_NAME_LENGTH</td>
<td>int</td>
<td>Number of characters in font of x &amp; y-axis labels; GetCtrlAttribute only.</td>
</tr>
<tr>
<td>ATTR_XYLABEL_ITALIC</td>
<td>int</td>
<td>1 = x &amp; y axes labels in italics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = x &amp; y axes labels not in italics</td>
</tr>
</tbody>
</table>

(continues)
For graphs and strip charts (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_XYLABEL_POINT_SIZE</td>
<td>int</td>
<td>Point size of x &amp; y axis names; range = 0 to 32767.</td>
</tr>
<tr>
<td>ATTR_XYLABEL_STRIKEOUT</td>
<td>int</td>
<td>1 = x &amp; y axes labels have strikeout</td>
</tr>
<tr>
<td>ATTR_XYLABEL_UNDERLINE</td>
<td>int</td>
<td>1 = x &amp; y axes labels underlined</td>
</tr>
<tr>
<td>ATTR_XYNAME_BOLD</td>
<td>int</td>
<td>1 = x &amp; y axes names bold</td>
</tr>
<tr>
<td>ATTR_XYNAME_COLOR</td>
<td>int</td>
<td>x- and y-axis name color (RGB value - see discussion that follows this table).</td>
</tr>
<tr>
<td>ATTR_XYNAME_FONT</td>
<td>char *</td>
<td>x- and y-axis name font (see Table 3-5).</td>
</tr>
<tr>
<td>ATTR_XYNAME_FONT_NAME_LENGTH</td>
<td>int</td>
<td>Number of characters in font of x &amp; y-axis names; GetCtrlAttribute only.</td>
</tr>
<tr>
<td>ATTR_XYNAME_ITALIC</td>
<td>int</td>
<td>1 = x &amp; y axes names in italics</td>
</tr>
<tr>
<td>ATTR_XYNAME_POINT_SIZE</td>
<td>int</td>
<td>Point size of x &amp; y axes names; range = 0 to 32767.</td>
</tr>
<tr>
<td>ATTR_XYNAME_STRIKEOUT</td>
<td>int</td>
<td>1 = x &amp; y axes names have strikeout</td>
</tr>
<tr>
<td>ATTR_XYNAME_UNDERLINE</td>
<td>int</td>
<td>1 = x &amp; y axes names underlined</td>
</tr>
<tr>
<td>ATTR_YDIVISIONS</td>
<td>int</td>
<td>Range: 1 to 100, or VAL_AUTO</td>
</tr>
<tr>
<td>ATTR_YENG_UNITS</td>
<td>int</td>
<td>Range: -308 to 308</td>
</tr>
<tr>
<td>ATTR_YFORMAT</td>
<td>int</td>
<td>See Table 3-12.</td>
</tr>
</tbody>
</table>

(continues)
For graphs and strip charts (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_YGRID_VISIBLE</td>
<td>int</td>
<td>1 = y-grid visible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = y-grid invisible</td>
</tr>
<tr>
<td>ATTR_YLABEL_VISIBLE</td>
<td>int</td>
<td>1 = y-label visible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = y-label invisible</td>
</tr>
<tr>
<td>ATTR_YMAP_MODE</td>
<td>int</td>
<td>VAL_LINEAR or VAL_LOG</td>
</tr>
<tr>
<td>ATTR_YNAME</td>
<td>char *</td>
<td>y-axis name</td>
</tr>
<tr>
<td>ATTR_YNAME_LENGTH</td>
<td>int</td>
<td>Number of characters in y-axis name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GetCtrlAttribute only.</td>
</tr>
<tr>
<td>ATTR_YPRECISION</td>
<td>int</td>
<td>Range: 0 to 15, or VAL_AUTO</td>
</tr>
</tbody>
</table>

For graphs only

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ACTIVE_YAXIS</td>
<td>int</td>
<td>Which of the two Y axes is used in plotting,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setting a Y axis attribute, setting the Y axis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>range, or creating a graph cursor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Values: VAL_LEFT_YAXIS,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VAL_RIGHT_YAXIS.</td>
</tr>
<tr>
<td>ATTR_COPY_ORIGINAL_DATA</td>
<td>int</td>
<td>See discussion below this table.</td>
</tr>
<tr>
<td>ATTR_DATA_MODE</td>
<td>int</td>
<td>VAL_RETAIN or VAL_DISCARD (see discussion that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>follows this table).</td>
</tr>
<tr>
<td>ATTR_ENABLE_ZOOMING</td>
<td>int</td>
<td>Whether the end-user can interactively zoom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and pan the graph viewport.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: FALSE</td>
</tr>
<tr>
<td>ATTR_NUM_CURSORS</td>
<td>int</td>
<td>Number of cursors (0 to 10).</td>
</tr>
<tr>
<td>ATTR_REFRESH_GRAPH</td>
<td>int</td>
<td>1 = plot to screen immediately</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = do not plot to screen until graph is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rescaled, overlapped, hidden, or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTR_REFRESH_GRAPH = 1</td>
</tr>
</tbody>
</table>

(continues)
## For graphs only (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_SHIFT_TEXT_PLOTS</td>
<td>int</td>
<td>1 = text is shifted so it will not be clipped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = text is shifted do it may be clipped</td>
</tr>
<tr>
<td>ATTR_SMOOTH_UPDATE</td>
<td>int</td>
<td>1 = use off-screen bitmap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = do not use off-screen bitmap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(See discussion that follows this table.)</td>
</tr>
<tr>
<td>ATTR_XMAP_MODE</td>
<td>int</td>
<td>VAL_LINEAR or VAL_LOG.</td>
</tr>
<tr>
<td>ATTR_XMARK_ORIGIN</td>
<td>int</td>
<td>1 = tick marks along x-origin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = no tick marks along x-origin</td>
</tr>
<tr>
<td>ATTR_XREVERSE</td>
<td>int</td>
<td>Whether to reverse the orientation of the X axis so that the lowest value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is shown at the right. If the orientation of the X axis is reversed, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>horizontal orientation of the plots is also reversed.</td>
</tr>
<tr>
<td>ATTR_YMARK_ORIGIN</td>
<td>int</td>
<td>1 = tick marks along y-origin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = no tick marks along y-origin</td>
</tr>
</tbody>
</table>

## For strip charts only

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_NUM_TRACES</td>
<td>int</td>
<td>Number of traces (1-64).</td>
</tr>
<tr>
<td>ATTR_POINTS_PER_SCREEN</td>
<td>int</td>
<td>Range: 3 to 10000</td>
</tr>
<tr>
<td>ATTR_SCROLL_MODE</td>
<td>int</td>
<td>VAL_SWEEP or VAL_CONTINUOUS or VAL_BLOCK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(See discussion that follows this table.)</td>
</tr>
</tbody>
</table>
For graph cursors (GetCursorAttribute and SetCursorAttribute)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CROSS_HAIR_STYLE</td>
<td>int</td>
<td>See Table 3-21.</td>
</tr>
<tr>
<td>ATTR_CURSOR_COLOR</td>
<td>int</td>
<td>Cursor color (RGB value—see discussion that follows this table.)</td>
</tr>
<tr>
<td>ATTR_CURSOR_MODE</td>
<td>int</td>
<td>VAL_FREE_FORM or VAL_SNAP_TO_POINT (See discussion that follows this table.)</td>
</tr>
<tr>
<td>ATTR_CURSOR_POINT_STYLE</td>
<td>int</td>
<td>See Table 3-22.</td>
</tr>
<tr>
<td>ATTR_CURSOR_YAXIS</td>
<td>int</td>
<td>Used to change the Y axis to which a graph cursor is associated. When you create a graph cursor, its associated Y axis is determined by the value of ATTR_ACTIVE_YAXIS. Afterwards, the association can be changed using ATTR_CURSOR_YAXIS. The associated axis serves as the reference for the cursor position coordinates used in calls to SetGraphCursor and GetGraphCursor. Values: VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS.</td>
</tr>
</tbody>
</table>

For graph plots (GetPlotAttribute and SetPlotAttribute) and for strip chart traces (GetTraceAttribute and SetTraceAttribute)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_LINE_STYLE</td>
<td>int</td>
<td>See Table 3-23.</td>
</tr>
<tr>
<td>ATTR_PLOT_STYLE</td>
<td>int</td>
<td>See Table 3-24.</td>
</tr>
<tr>
<td>ATTR_TRACE_COLOR</td>
<td>int</td>
<td>Trace color (RGB value—see discussion that follows this table).</td>
</tr>
<tr>
<td>ATTR_TRACE_POINT_STYLE</td>
<td>int</td>
<td>See Table 3-22.</td>
</tr>
<tr>
<td>ATTR_TRACE_VISIBLE</td>
<td>int</td>
<td>1 = trace is visible; 0 = trace is invisible</td>
</tr>
</tbody>
</table>
### For graph plots (GetPlotAttribute and SetPlotAttribute)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_INTERPOLATE_PIXELS</td>
<td>int</td>
<td>Enable the calculation of the displayed color for each pixel by using interpolation. n 0 = Interpolate pixels 1 = No Interpolation (Valid for Intensity plots only.)</td>
</tr>
<tr>
<td>ATTR_NUM_POINTS</td>
<td>int</td>
<td>Number of points in the plot data. For intensity plots, the number of points will be equal to the number of points in the 2 dimensional z-data array (valid for X, Y, XY, Waveform, Polygon and Intensity plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_FONT</td>
<td>char *</td>
<td>The font name for the text plot (valid for PlotText plots only).</td>
</tr>
<tr>
<td>ATTR_PLOT_FONT_NAME_LENGTH</td>
<td>int</td>
<td>The length of the font name for the text plot (valid for PlotText plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_ORIGIN</td>
<td>int</td>
<td>Determines the placement of a text string or bitmap with respect to the coordinates specified in a call to PlotText or PlotBitmap. See discussion following this table.</td>
</tr>
<tr>
<td>ATTR_PLOT_SNAPPABLE</td>
<td>int</td>
<td>By default, graph cursors for which ATTR_CURSOR_MODE is VAL_SNAP_TO_POINT snap to the closest plot. To prevent cursors from snapping to a particular plot, set ATTR_PLOT_SNAPPABLE for the plot to FALSE.</td>
</tr>
<tr>
<td>ATTR_PLOT_YAXIS</td>
<td>int</td>
<td>Used to change the Y axis with which a plot is associated. When a plot is first plotted, the Y axis to which it is associated is determined by the value of ATTR_ACTIVE_YAXIS. Afterwards, the association can be changed using ATTR_PLOT_YAXIS. Values: VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS.</td>
</tr>
</tbody>
</table>

(continues)
For graph plots (GetPlotAttribute and SetPlotAttribute) (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_PLOT_ZPLANE_POSITION</td>
<td>int</td>
<td>The drawing order of the graph plot. The lowest ordered plot(0) is on top. Valid Range: 0 to (Number of Plots - 1)</td>
</tr>
<tr>
<td>ATTR_TRACE_BGCOLOR</td>
<td>int</td>
<td>The RGB color value for the background text of a text plot or the fill color for drawn object type plot (valid for Text, Rectangle, Polygon, Oval and Arc plots only).</td>
</tr>
<tr>
<td>ATTR_PLOT_XDATA</td>
<td>void *</td>
<td>A void pointer to a buffer for the X data to be copied to. (Valid for X, XY and Polygon plots only.) (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_YDATA</td>
<td>void *</td>
<td>A void pointer to a buffer for the Y data to be copied to (valid for Y, XY, Waveform and Polygon plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_ZDATA</td>
<td>void *</td>
<td>A void pointer to a buffer for the Z data to be copied to. This buffer will be a two-dimensional array (valid for Intensity plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_XDATA_TYPE</td>
<td>int</td>
<td>The type of data in the X data plot (valid for X, XY and Polygon plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_YDATA_TYPE</td>
<td>int</td>
<td>The type of data in the Y data plot (valid for Y, XY, Waveform and Polygon plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_ZDATA_TYPE</td>
<td>int</td>
<td>The type of data in the Z data plot (valid for Intensity plots only). (GetPlotAttribute only.)</td>
</tr>
</tbody>
</table>

(continues)
For graph plots (GetPlotAttribute and SetPlotAttribute) (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_PLOT_XDATA_SIZE</td>
<td>int</td>
<td>The number of bytes in the X data plot data (valid for X, XY and Polygon plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_YDATA_SIZE</td>
<td>int</td>
<td>The number of bytes in the Y data plot data (valid for Y, XY, Waveform and Polygon plots only). (GetPlotAttribute only.)</td>
</tr>
<tr>
<td>ATTR_PLOT_ZDATA_SIZE</td>
<td>int</td>
<td>The number in bytes in the Z data plot data (valid for Intensity plots only). (GetPlotAttribute only.)</td>
</tr>
</tbody>
</table>

Graph Attribute Discussion

An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. You can use the User Interface Library function MakeColor to create an RGB value from red, green, and blue color components. See Table 3-3 for a list of common color values.

The ATTR_DATA_MODE attribute controls whether LabWindows/CVI keeps or discards the scaled plot data for subsequent plots after drawing the data to the screen. If you set the attribute to VAL_RETAIN, LabWindows/CVI retains scaled plot data in memory and accesses the data when the plot is overlapped or hidden. If you set the attribute to VAL_DISCARD, the scaled plot data is freed from memory as soon as the plot is drawn. This conserves memory, but plots are lost if the graph is rescaled, overlapped, or hidden.

The ATTR_COPY_ORIGINAL_DATA attribute specifies whether LabWindows/CVI makes a copy of the original plot data for each new plot added to the graph. The original plot data is needed whenever the graph is rescaled and the original plot data has been overwritten in your program.

The ATTR_SMOOTH_UPDATE attribute specifies whether LabWindows/CVI stores a copy of the graph in an off-screen bitmap. Using an off-screen bitmap results in less plot flicker and smoother cursor movement, but consumes more memory.

The ATTR_SCROLL_MODE attribute specifies the scrolling mode of the strip chart. If you set the attribute to VAL_CONTINUOUS, old data scrolls off the left edge of the plot area as new data plots at the right edge. If you set the attribute to VAL_Sweep, new data overwrites old data from left to right. If you set the attribute to VAL_BLOCK, the entire plot area is erased when data reaches the right edge of the area.
The `ATTR_CURSOR_MODE` specifies the behavior of a graph cursor. You can move `VAL_FREE_FORM` cursors to any location inside the plot area. `VAL_SNAP_TO_POINT` cursors snap to the nearest data point when released.

Table 3-21 shows the cursor styles associated with `ATTR_CROSS_HAIR_STYLE`. (Assume that `ATTR_CURSOR_POINT_STYLE` is set to `VAL_SIMPLE_DOT`.)

<table>
<thead>
<tr>
<th>Value</th>
<th>Cross hair Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_LONG_CROSS</td>
<td>——</td>
</tr>
<tr>
<td>VAL_VERTICAL_LINE</td>
<td></td>
</tr>
<tr>
<td>VAL_HORIZONTAL_LINE</td>
<td>——</td>
</tr>
<tr>
<td>VAL_NO_CROSS</td>
<td></td>
</tr>
<tr>
<td>VAL_SHORT_CROSS</td>
<td>——</td>
</tr>
</tbody>
</table>

Table 3-22 shows the styles associated with `ATTR_CURSOR_POINT_STYLE` and `ATTR_TRACE_POINT_STYLE`.

Table 3-22. Styles for `ATTR_CURSOR_POINT_STYLE` and `ATTR_TRACE_POINT_STYLE`

<table>
<thead>
<tr>
<th>Value Style Name</th>
<th>Cursor/Point Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_EMPTY_SQUARE</td>
<td>□</td>
</tr>
<tr>
<td>VAL_SOLID_SQUARE</td>
<td>■</td>
</tr>
<tr>
<td>VAL_ASTERISK</td>
<td>★</td>
</tr>
<tr>
<td>VAL_DOTTED_EMPTY_SQUARE</td>
<td>◆</td>
</tr>
<tr>
<td>VAL_DOTTED_SOLID_SQUARE</td>
<td>◆</td>
</tr>
<tr>
<td>VAL_SOLID_DIAMOND</td>
<td>◆</td>
</tr>
<tr>
<td>VAL_EMPTY_SQUARE_WITH_X</td>
<td>◆</td>
</tr>
<tr>
<td>VAL_EMPTY_SQUARE_WITH_CROSS</td>
<td>◆</td>
</tr>
<tr>
<td>VAL_BOLD_X</td>
<td>★</td>
</tr>
<tr>
<td>VAL_SMALL_SOLID_SQUARE</td>
<td>□</td>
</tr>
</tbody>
</table>

(continues)
Table 3-22. Styles for ATTR_CURSOR_POINT_STYLE and ATTR_TRACE_POINT_STYLE (Continued)

<table>
<thead>
<tr>
<th>Value Style Name</th>
<th>Cursor/Point Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_SIMPLE_DOT</td>
<td>.</td>
</tr>
<tr>
<td>VAL_EMPTY_CIRCLE</td>
<td>●</td>
</tr>
<tr>
<td>VAL_SOLID_CIRCLE</td>
<td>●●</td>
</tr>
<tr>
<td>VAL_DOTTED_SOLID_CIRCLE</td>
<td>●●●</td>
</tr>
<tr>
<td>VAL_DOTTED_EMPTY_CIRCLE</td>
<td>○●●</td>
</tr>
<tr>
<td>VAL_BOLD_CROSS</td>
<td>+</td>
</tr>
<tr>
<td>VAL_CROSS</td>
<td>+</td>
</tr>
<tr>
<td>VAL_SMALL_CROSS</td>
<td>+●</td>
</tr>
<tr>
<td>VAL_X</td>
<td>✗</td>
</tr>
<tr>
<td>VAL_SMALL_X</td>
<td>✗●</td>
</tr>
<tr>
<td>VAL_DOTTED_SOLID_DIAMOND</td>
<td>●♦</td>
</tr>
<tr>
<td>VALEMPTY_DIAMOND</td>
<td>●♦</td>
</tr>
<tr>
<td>VAL_DOTTED_EMPTY_DIAMOND</td>
<td>●♦</td>
</tr>
<tr>
<td>VAL_SMALL_EMPTY_SQUARE</td>
<td>●♦</td>
</tr>
<tr>
<td>VAL_NO_POINT</td>
<td></td>
</tr>
</tbody>
</table>

Table 3-23 shows the line styles associated with ATTR_LINE_STYLE. (Assume that ATTR_PLOT_STYLE is set to VAL_THIN_LINE.)

Table 3-23. Line Styles for ATTR_LINE_STYLE

<table>
<thead>
<tr>
<th>Value</th>
<th>Line Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_SOLID</td>
<td></td>
</tr>
<tr>
<td>VAL_DASH</td>
<td>/</td>
</tr>
<tr>
<td>VAL_DOT</td>
<td>/</td>
</tr>
<tr>
<td>VAL_DASH_DOT</td>
<td>/</td>
</tr>
<tr>
<td>VAL_DASH_DOT_DOT</td>
<td>/</td>
</tr>
</tbody>
</table>
Table 3-24 shows the plot styles associated with ATTR_PLOT_STYLE. (Assume that the point style is set to VAL_ASTERISK.)

### Table 3-24. Plot Styles for ATTR_PLOT_STYLE

<table>
<thead>
<tr>
<th>Value</th>
<th>Plot Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_THIN_LINE</td>
<td><img src="image" alt="Thin Line" /></td>
</tr>
<tr>
<td>VAL_FAT_LINE</td>
<td><img src="image" alt="Fat Line" /></td>
</tr>
<tr>
<td>VAL_CONNECTED_POINTS</td>
<td><img src="image" alt="Connected Points" /></td>
</tr>
<tr>
<td>VAL_SCATTER</td>
<td><img src="image" alt="Scatter" /></td>
</tr>
<tr>
<td>VAL_THIN_STEP</td>
<td><img src="image" alt="Thin Step" /></td>
</tr>
<tr>
<td>VAL_FAT_STEP</td>
<td><img src="image" alt="Fat Step" /></td>
</tr>
<tr>
<td>VAL_VERTICAL_BAR</td>
<td><img src="image" alt="Vertical Bar" /> (not valid for strip charts)</td>
</tr>
<tr>
<td>VAL_HORIZONTAL_BAR</td>
<td><img src="image" alt="Horizontal Bar" /> (not valid for strip charts)</td>
</tr>
<tr>
<td>VAL_BASE_ZERO_VERTICAL_BARS</td>
<td><img src="image" alt="Base Zero Vertical Bars" /> (not valid for strip charts)</td>
</tr>
<tr>
<td>VAL_BASE_ZERO_HORIZONTAL_BARS</td>
<td><img src="image" alt="Base Zero Horizontal Bars" /> (not valid for strip charts)</td>
</tr>
</tbody>
</table>

**Note:** Under Windows, the plot style VAL_FAT_LINE forces the line style to be VAL_SOLID.

### Plot Origin Discussion

When PlotText or PlotBitmap is called, the text string or bitmap is placed on the graph with respect to a point specified by coordinates passed into the function. The orientation of the string or bitmap with respect to the point is determined by the ATTR_PLOT_ORIGIN attribute. The attribute specifies where the point (that is, the origin) is with respect to the rectangle that implicitly encloses the string or bitmap. For example, VAL_LOWER_LEFT (the default value) specifies that the string or bitmap be plotted so that the lower left corner of its enclosing rectangle is at the point specified.
The possible values are shown in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_LOWER_LEFT</td>
<td>The lower left corner of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_CENTER_LEFT</td>
<td>The midpoint of the left edge of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_UPPER_LEFT</td>
<td>The upper left corner of the enclosing rectangle.</td>
</tr>
<tr>
<td>VALLOWER_CENTER</td>
<td>The midpoint of the bottom edge of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_CENTER_CENTER</td>
<td>The center of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_UPPER_CENTER</td>
<td>The midpoint of the top edge of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_LOWER_RIGHT</td>
<td>The lower right corner of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_CENTER_RIGHT</td>
<td>The midpoint of the right edge of the enclosing rectangle.</td>
</tr>
<tr>
<td>VAL_UPPER_RIGHT</td>
<td>The upper right corner of the enclosing rectangle.</td>
</tr>
</tbody>
</table>

**Two Y Axis (graphs only)**

There are always two Y axes on a graph. By default, only the left Y axis is visible.

You can make the Y axis visible by using the following code.

```c
SetCtrlAttribute (panel, ctrl, ATTR_ACTIVE_YAXIS, VAL_RIGHT_AXIS);
SetCtrlAttribute (panel, ctrl, ATTR_YLABEL_VISIBLE, 1);
```

You can choose to make either one, both, or none of the Y axes visible.

The ATTR_ACTIVE_YAXIS attribute determines which of the two Y axes are used for the following actions.

- Adding a plot to the graph. (The active Y axis serves as the scaling reference.)
- Setting a Y axis attribute. (Each Y axis has its own attribute values.)
- Setting the Y axis range.
- Creating a graph cursor. (The cursor is associated with the active Y axis.)

Once a plot has been added to a graph, you can associate it with the other Y axis by using the ATTR_PLOT_YAXIS attribute.

Once a graph cursor has been created, you can associate it with the other Y axis by using the ATTR_CURSOR_YAXIS attribute. The associated Y axis serves as the reference for the cursor position coordinates in calls to SetGraphCursor and GetGraphCursor.
Optimizing Graph Controls

This section presents attributes related to graphing and discusses how various settings affect performance.

Optimizing Speed

The following attributes affect the speed and appearance of your graph displays as they update.

Speed and ATTR_SMOOTH_UPDATE

When you enable ATTR_SMOOTH_UPDATE or select Smooth Update in the Graph control editor, your graph updates first to an off-screen buffer and then to the screen.

ATTR_SMOOTH_UPDATE improves performance as follows.

- Eliminates flashing associated with plotting large plots.
- Optimizes speed when you use graph cursors.
- Optimizes speed when a graph is updated after it has been overlapped or hidden.

ATTR_SMOOTH_UPDATE has the following disadvantages.

- Initial plotting speed is slower.
- Memory usage increases.

Smooth updating of the graph is automatically disabled when you make the color of the plot background transparent or when a visible item overlaps the plot area.

ATTR_SMOOTH_UPDATE slows the initial graphing of a plot because a graph updates to the offscreen buffer before plotting to the screen. However, when another window is covering a plot, and then you remove that window, ATTR_SMOOTH_UPDATE lets LabWindows/CVI refresh a graph quickly because the graph already exists in the offscreen buffer.

Speed and VAL_AUTO_SCALE

When you enable autoscaling both the recalculation of the axes limits and the remapping of graph plots are dynamically adjusted with every new plot. As the number of existing plots on a graph increase, you need to try to minimize scaling adjustments because the time needed to recalculate and remap all of the existing plots increases. You may want to minimize delays by disabling autoscaling, which prevents the recalculation and remapping of the graph plots. To disable autoscaling, call the SetAxisRange function with the xAxisScaling and the yAxisScaling parameters each set to VAL_MANUAL or VAL_LOCK.
Controlling How Graphs Refresh

You can use the **refresh** parameter of the `DeleteGraphPlot` function and the `RefreshGraph` function to determine when a graph is updated. A “draw event” signals LabWindows/CVI to refresh its graph. The last parameter of `DeleteGraphPlot` determines when a draw event is generated. To delete plots from a graph, your program can call `DeleteGraphPlot` with **refresh** set to **VAL_DELAYED_DRAW**, so that plots do not disappear until a draw event occurs.

The following conditions generate draw events.

- A GUI object covers the plot area.
- Almost any graph attribute changes.
- Adding a new plot while **REFRESH_GRAPH** is enabled.
- `DeleteGraphPlot` is called with the **refresh** parameter set to **VAL_IMMEDIATE_DRAW**.
- The program enables **REFRESH_GRAPH** through `SetCtrlAttribute`.
- The cursor moves over the graph.
- The program calls `RefreshGraph`.
- A call to `ProcessDrawEvents` or `ProcessSystemEvents` updates graph attribute changes. If a plot update is pending as well, the attribute change makes that plot update, too.

The preferable way to delete a plot and then plot a new one is to set **refresh** to **VAL_DELAYED_DRAW** when deleting, then plot the new plot. When you do this, and at the same time you enable smooth updating, you eliminate screen flashing and improve the speed of the update.

`DeleteGraphPlot` deletes all plots on a graph when the **plotHandle** parameter is set to -1.

**ATTR_REFRESH_GRAPH** determines whether your program plots on a graph control immediately or plots after the graph is rescaled, overlapped, hidden or the attribute is enabled.

When your program builds a multi-plot image, for which only the final appearance is of interest, the attribute should be turned off before plotting begins, and turned on when it ends. This causes only one draw event to occur, instead of one for each plot. Plotting occurs even more rapidly when you enable the smooth updates attribute. Plots that you add while **REFRESH_GRAPH** is disabled are held in **pending** status: they are part of the graph, but are invisible until a draw event takes place. Pending plots only appear after a call to `RefreshGraph` or when the **ATTR_REFRESH_GRAPH** attribute is enabled.
Optimizing Memory Usage

You can optimize memory usage by disabling the ATTR_COPY_ORIGINAL_DATA attribute and the ATTR_DATA_MODE setting, but the savings are not significant in comparison to the overall amount of memory that LabWindows/CVI uses.

The ATTR_SMOOTH_UPDATE graph attribute saves an off-screen bitmap of the graph. You can save memory by disabling this attribute.

The ATTR_DATA_MODE graph attribute lets you retain or discard scaled plot data for subsequent plots.

- When you disable ATTR_SMOOTH_UPDATE and set ATTR_DATA_MODE to discard, you get the following results.
  - No data prints with PrintPanel or PrintCtrl.
  - Plot data is lost when the graph is rescaled, hidden, overlapped or when a draw event occurs. (Reasons for rescaling are listed later in this subsection.)

- When you enable ATTR_SMOOTH_UPDATE and set ATTR_DATA_MODE to discard, you get the following result.
  - Plot data is lost when the graph is rescaled or when a draw event occurs.

Note: You cannot delete plots that you add to a graph while ATTR_DATA_MODE is set to discard because the plot functions do not return a plot ID. The graph behaves as if the plot does not exist.

Your graph can rescale for the following reasons.

- SetAxisRange is called.

- ATTR_XMAP_MODE or ATTR_YMAP_MODE changes.

- You draw a new plot when autoscaling is enabled and the minimum or maximum values change.

- You change graph attributes that affect the size of the plot area.

The ATTR_COPY_ORIGINAL_DATA graph attribute determines whether the graph control will keep its own copy of the plot data. When disabled, the graph control maintains a pointer to the original array data used that was used when you originally made a call to that plotting function. If the data in the original array changes or is deallocated, either the graph displays incorrect data or an invalid pointer error occurs. ATTR_COPY_ORIGINAL_DATA only affects graph plots that are associated with arrays, such as PlotWaveform and PlotXY. The graph control only needs to go back to the original data when the graph is rescaled.
Programming with Pop-Up Panels

Use InstallPopup to display and activate a panel as a dialog box. You must load a panel with the LoadPanel function or create the panel using the NewPanel function.

After a pop-up panel is installed, users can perform operations in LabWindows/CVI only on the pop-up panel.

Only the active pop-up panel can generate events (with the exception EVENT_PANEL_MOVE, EVENT_PANEL_SIZE, and EVENT_CLOSE events from other panels). Use callback functions to process any kind of event or GetUserEvent to only process commit events. GetUserEvent returns the ID of the control that caused the event. GetUserEvent can operate in one of two ways.

• GetUserEvent waits for the user to generate an event before returning to the calling program.

• GetUserEvent returns immediately whether or not an event has occurred.

Use RemovePopup to remove either the active pop-up panel or all pop-up panels. RemovePopup does not unload the panel from memory.

Use SetSystemPopupAttributes and GetSystemPopupAttributes to set or obtain the values of attributes that affect all of the pop-up panel. The predefined pop-up panels are accessed through the following functions.

ConfirmPopup
DirSelectPopup
FileSelectPopup
FontSelectPopup
GenericMessagePopup
MessagePopup
MultifileSelectPopup
PromptPopup
SetFontPopupDefaults
WaveformGraphPopup
XGraphPopup
XYGraphPopup
YGraphPopup

These functions handle the installation, user interaction with, and removal of the pop-up panels.
Using the System Attributes

The system attributes are set and obtained using the `SetSystemAttribute` and `GetSystemAttribute` functions. They are attributes that apply to the User Interface Library in general, rather than to particular instances of user interface objects. The following table lists the system attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ALLOW_UNSAFE_TIMER_EVENTS</td>
<td>int</td>
<td>1 - Allow unsafe timer events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - Do not allow unsafe time events (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Windows 95 and NT only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See discussion below.</td>
</tr>
<tr>
<td>ATTR_ALLOW_MISSING_CALLBACKS</td>
<td>int</td>
<td>1 - <code>LoadPanel</code> and <code>LoadMenuBar</code> return a valid panel or menu bar handle even if not all callback functions referenced in the .uir file can be found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - <code>LoadPanel</code> and <code>LoadMenuBar</code> return an error code if not all callback functions referenced in the .uir file can be found (the default).</td>
</tr>
<tr>
<td>ATTR_REPORT_LOAD_FAILURE</td>
<td>int</td>
<td>1 - Display a message when <code>LoadPanel</code> or <code>LoadMenuBar</code> fail (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - Do not display a message when <code>LoadPanel</code> or <code>LoadMenuBar</code> fail See discussion below.</td>
</tr>
<tr>
<td>ATTR_SUPPRESS_EVENT_PROCESSING</td>
<td>int</td>
<td>1 - No events are processed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - Events are processed normally (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See discussion below.</td>
</tr>
</tbody>
</table>

Unsafe Timer Events

By default, timer control callbacks do not occur on Windows while you are moving or sizing a window, while the system menu is pulled down, or while the Alt-Tab key is pressed. (These conditions are called *event-blocking conditions*.) On Windows 95 and NT, you can use the `ALLOW_UNSAFE_TIMER_EVENTS` attribute to enable timer events under some, but not all, of the event-blocking conditions. If you set the `ALLOW_UNSAFE_TIMER_EVENTS` attribute to TRUE, timer events are blocked only under the following conditions.
• You have clicked on a window title bar, you are holding the mouse button down, but you are not moving the mouse.

• You are moving or resizing a window, and the Windows 95 “Show Window Contents While Dragging” option is disabled (or you are running on Windows NT.)

There are several limitations to this feature. One limitation is that while an event-blocking condition is in effect, timer callbacks are called no faster than once per 55 milliseconds.

Another limitation is that if a timer callback is called during an event-blocking condition and the callback causes events to be processed, mouse and keyboard input can behave erratically. Your program can cause this to happen in the following ways.

• The timer callback function calls ProcessSystemEvents in a loop.

• The timer callback function calls RunUserInterface, GetUserEvent, or a popup panel function such as MessagePopup or FileSelectPopup.

• Program execution is suspended in the timer callback function because of a breakpoint or run-time error.

In either case, the system functions normally once the timer callback returns.

This problem is inherent to Windows and occurs regardless of the development environment.

You should not enable this attribute until the code in your timer callbacks has been thoroughly debugged. The behavior of the system is undefined if you hit a breakpoint or run-time error when an event-blocking condition is in effect.

Reporting Load Failures

The LoadPanel, LoadPanelEx, LoadMenuBar, and LoadMenuBarEx functions can fail for many reasons. For instance, the library may not be able to find the .uir file. Or a callback function specified in the panel or menu bar may not be defined in the project, executable, or DLL.

It is not unusual for a load functions to fail in a standalone executable even though it succeeded in LabWindows/CVI. If you do not check for errors in your program, the behavior of the program can be very confusing. ATTR_REPORT_LOAD_FAILURE, when enabled, causes an error message to be displayed when one of the load functions fails. The error message is displayed in all cases except when debugging and the Break on Library Errors feature are in effect. The attribute is enabled by default.

If a load function fails in a standalone executable or DLL created in an external compiler, the most common reasons are:

• The executable was created in a directory different than where the LabWindows/CVI project file was saved. This can cause the library to fail to find the .uir file.
• You did not create a .UIR Callbacks Object File, or you did not add it to your external compiler project. (See the External Compiler Support command in the Build menu of the Project Window.) This will cause the library to fail to find callback functions referenced in the .uir file.

• You did not call InitCVIRTE at the beginning of your program or in DLLMain. This will cause the library to fail to find callback functions referenced in the .uir file.

• Your callback functions are in a DLL but are not exported by the DLL, and you are not using LoadPanelEx or LoadMenuBarEx. This will cause the library to fail to find callback functions referenced in the .uir file.

Suppressing Event Processing

If you call QuitUserInterface to terminate a RunUserInterface call and there are already events in the event queue, RunUserInterface processes these events before it terminates. To ensure that no events are processed and that no callbacks are called before RunUserInterface terminates, set the ATTR_SUPPRESS_EVENT_PROCESSING attribute to 1. This attribute also suppresses event processing and callbacks if it is enabled while a call to GetUserEvent or ProcessSystemEvents is in effect. The attribute is automatically reset to zero when another call to RunUserInterface, GetUserEvent, or ProcessSystemEvents is made.

Generating Hard Copy Output

You can generate hard copy output of panels or individual controls to a graphics printer or a file.

Functions for Hard Copy Output

Use PrintPanel to output a panel and its child panels.

Use PrintCtrl to output an individual control.

Use SetPrintAttribute to set hard copy attributes.

Use GetPrintAttribute to obtain hard copy attributes.
### Hard Copy Attributes

Table 3-27 contains the list of control attributes accessible through `GetPrintAttribute` and `SetPrintAttribute`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_COLOR_MODE</td>
<td>int</td>
<td>0 = VAL_BW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = VAL_GRAY_SCALE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = VAL_COLOR</td>
</tr>
<tr>
<td>ATTR_DUPLEX</td>
<td>int</td>
<td>1 = VAL_SIMPLEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = VAL_VERTDUPLEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = VAL_HORIZDUPLEX</td>
</tr>
<tr>
<td>ATTR_EJECT_AFTER</td>
<td>int</td>
<td>1 = eject page after print</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = do not eject page after print</td>
</tr>
<tr>
<td>ATTR_NUMCOPIES</td>
<td>int</td>
<td>Number of copies; range = 1 to 100</td>
</tr>
<tr>
<td>ATTR_ORIENTATION</td>
<td>int</td>
<td>1 = VAL_PORTrait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = VAL_LANDSCAPE</td>
</tr>
<tr>
<td>ATTR_PAPER_HEIGHT</td>
<td>int</td>
<td>Millimeter/10 or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 = VAL_USE_PRINTER_DEFAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = VAL_INTEGRAL_SCALE</td>
</tr>
<tr>
<td>ATTR_PAPER_WIDTH</td>
<td>int</td>
<td>Millimeter/10 or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 = VAL_USE_PRINTER_DEFAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = VAL_INTEGRAL_SCALE</td>
</tr>
<tr>
<td>ATTR_TAB_INTERVAL</td>
<td>int</td>
<td>The number of spaces represented by a &lt;Tab&gt; character.</td>
</tr>
<tr>
<td>ATTR_TEXT_WRAP</td>
<td>int</td>
<td>Determines whether to wrap text when text extends past the defined width.</td>
</tr>
<tr>
<td>ATTR_XOFFSET</td>
<td>int</td>
<td>Inches or VAL_USE_PRINTER_DEFAULT</td>
</tr>
<tr>
<td>ATTR_YOFFSET</td>
<td>int</td>
<td>Inches or VAL_USE_PRINTER_DEFAULT</td>
</tr>
<tr>
<td>ATTR_XRESOLUTION</td>
<td>int</td>
<td>Dots Per Inch or VAL_USE_PRINTER_DEFAULT</td>
</tr>
<tr>
<td>ATTR_YRESOLUTION</td>
<td>int</td>
<td>Dots Per Inch or VAL_USE_PRINTER_DEFAULT</td>
</tr>
</tbody>
</table>
Hard Copy Attribute Discussion

The following list shows attribute names and their purpose.

**ATTR_DUPLICATE** — Determines if the output is single or double sided.

**ATTR_EJECT_AFTER** — Determines if the next output is ejected from the printer. While **ATTR_EJECT_AFTER** is set to zero, outputs print on the same page until **ATTR_EJECT_AFTER** is set to one.

**ATTR_ORIENTATION** — Determines if the hard copy is in portrait or landscape mode.

**ATTR_PAPER_HEIGHT** — **VAL_INTEGRAL_SCALE** forces the hard copy height to be scaled integrally to the width preventing printer aliasing (distortion).

**ATTR_PAPER_WIDTH** — **VAL_INTEGRAL_SCALE** forces the hard copy width to be scaled integrally to the height preventing printer aliasing (distortion).

**ATTR_XOFFSET** — Sets the x offset of the hard copy image on the paper. The coordinates (0,0) define the upper-left corner of the paper. **VAL_USE_PRINTER_DEFAULT** centers the image in the x direction.

**ATTR_YOFFSET** — Sets the y offset of the hard copy image on the paper. The coordinates (0,0) define the upper-left corner of the paper. **VAL_USE_PRINTER_DEFAULT** centers the image in the y direction.

**ATTR_XRESOLUTION** — Sets the x resolution of the printer. It can be used if the printer supports different resolutions in the x and y dimensions.

**ATTR_YRESOLUTION** — Sets the y resolution of the printer. It can be used if the printer supports different resolutions in the x and y dimensions.

**Note:** Because of a limitation in the UNIX xpr printing utility, UNIX users cannot print multiple outputs on the same page when their printer is configured for Postscript.
Table 3-28. Values for ATTR_COLOR_MODE

<table>
<thead>
<tr>
<th>Type of Printer</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>default value</td>
<td>VAL_COLOR</td>
</tr>
<tr>
<td><strong>PC with color printer:</strong></td>
<td></td>
</tr>
<tr>
<td>VAL_COLOR: prints in color</td>
<td></td>
</tr>
<tr>
<td>VAL_GRAYSCALE: prints in grayscale</td>
<td></td>
</tr>
<tr>
<td>VAL_BW: undefined</td>
<td></td>
</tr>
<tr>
<td><strong>PC with non-color printer</strong></td>
<td></td>
</tr>
<tr>
<td>VAL_COLOR: prints in grayscale</td>
<td></td>
</tr>
<tr>
<td>VAL_GRAYSCALE: prints in grayscale</td>
<td></td>
</tr>
<tr>
<td>VAL_BW: prints in black and white</td>
<td></td>
</tr>
<tr>
<td><strong>Sun with color printer</strong></td>
<td></td>
</tr>
<tr>
<td>VAL_COLOR: prints in color</td>
<td></td>
</tr>
<tr>
<td>VAL_GRAYSCALE: prints in grayscale</td>
<td></td>
</tr>
<tr>
<td>VAL_BW: undefined</td>
<td></td>
</tr>
<tr>
<td><strong>Sun with non-color printer</strong></td>
<td></td>
</tr>
<tr>
<td>VAL_COLOR: undefined</td>
<td></td>
</tr>
<tr>
<td>VAL_GRAYSCALE: prints in grayscale</td>
<td></td>
</tr>
<tr>
<td>VAL_BW: prints in black and white</td>
<td></td>
</tr>
</tbody>
</table>

**Special User Interface Functions**

**RunUserInterface**

RunUserInterface runs the GUI and issues all events to callback functions, until QuitUserInterface is called from a callback. The return value for RunUserInterface is passed back from QuitUserInterface.

**Precedence of Callback Functions**

Events trigger callback functions in the following order.

- For control operation events:
  1. Control callback
  2. Panel callback (keypress and mouse events only)
  3. Main callback
• For panel events:
  1. Panel callback
  2. Main callback

• For menu commit events:
  1. Menu item callback
  2. Main callback

• Timer control event:
  1. Control callback

• Main callback event:
  1. Main Callback

Note: The commit event is placed in the `GetUserEvent` queue after being sent to all callbacks.

Swallowing Events

User callbacks must always return 0 unless they intend to "swallow" the event to which they are responding. To swallow the event, the callback should return 1.

Only user input (mouse click and keypress events) and commit events can be swallowed. If swallowed, no further callbacks are called for that event. If a user input event is swallowed, the user’s mouse click or keypress is ignored. If a commit event is swallowed, it is not placed into the `GetUserEvent` queue.

Note: The events that can be swallowed are:

- `EVENT_COMMIT`
- `EVENT_KEYPRESS`
- `EVENT_LEFT_CLICK`
- `EVENT_LEFT_DOUBLE_CLICK`
- `EVENT_RIGHT_CLICK`
- `EVENT_RIGHT_DOUBLE_CLICK`
- `EVENT_END_TASK`

GetUserEvent

`GetUserEvent` gets the next commit event from the `GetUserEvent` queue. Commit events occur when the user changes the state of a hot or validate control or selects a menu item. You can set `GetUserEvent` to wait until the next event or to return immediately even if no event occurs.

InstallMainCallback and SetIdleEventRate

An object-based callback approach is a common method of using callbacks to process user events. In an object-based callback approach, you use multiple callbacks to control user interface objects. You can assign unique callback functions to each panel, control, and menu item, or you can use separate callback functions for specific groups of controls or menu items.
You can use any scheme that makes sense to you. The object-based callback approach subdivides the program into small, manageable sections, each section having a specific task.

Alternatively, you can install a single callback function using `InstallMainCallback` to process all events. A main callback is like a `GetUserEvent` loop that is “turned upside-down.”

The main callback function is the only type of callback function that responds to `EVENT_END_TASK`. This event occurs only on MS Windows when the user is trying to exit Windows. Return a non-zero value to abort the exit of Windows.

The main callback function also responds to idle events. For example, your program will normally be suspended in the `GetUserEvent` or `RunUserInterface` function while a user holds the mouse button down on a control or pull-down menu. Other than timer control callbacks, any portion of your program that is supposed to run continuously temporarily ceases during that time. However, idle events occur continuously while your program is running, even while suspended in `GetUserEvent` or `RunUserInterface`. You can set the rate at which idle events occur using `SetIdleEventRate`.

Note: `EVENT_VAL_CHANGED`, `EVENT_IDLE` and `EVENT_TIMER_TICK` are the only events that are generated while a user holds the mouse button down on a control or pull-down menu. The operating system blocks all events (including idle events) when a top-level panel is moved or sized.

Note: In general, it is recommended that you use timer controls instead of idle events.

For even greater flexibility, you can combine the object-based callback approach with the main callback approach. Object-based callbacks will be called before the main callback so that both have the opportunity to process events.

**ProcessDrawEvents**

While inside of a callback function or in code that does not call `RunUserInterface` or `GetUserEvent`, the program does not update the user interface. If a particular function is overly time-consuming, it essentially “locks out” user interface updates. To force these updates to be processed, place a call to `ProcessDrawEvents` in your source code.

Note: The user interface is updated automatically by the `GetUserEvent` function and when a callback returns.

**ProcessSystemEvents**

While a callback function or other section of code that does not call `RunUserInterface` or `GetUserEvent` runs, user interface and system events are not processed. While such functions or code run, they temporarily stop user interface and system events. The `ProcessSystemEvents` function lets you force events to process while a time-consuming
function is running. You can make your program call `ProcessSystemEvents` occasionally, from within the function that is preventing system events. Because `ProcessSystemEvents` can allow other callback functions to be executed before it completes, you must use this function with care.

The `ProcessSystemEvents` function processes all pending system events, such as,

- System events that are delayed or suspended by a user application, for example keystrokes, mouse events, and screen updates.
- Events generated by other applications, for example, Windows messages intended to invoke a callback within the `RegisterWinMsgCallback` function.

**Note:** The `ProcessSystemEvents` function is called automatically by the `LabWindows/CVI GetUserEvent` function and after a callback returns.

**PostDeferredCall**

`PostDeferredCall` posts a function to LabWindows/CVI that is called at the next occurrence of `GetUserEvent`, `RunUserInterface`, or `ProcessSystemEvents`. `PostDeferredCall` would typically be used in a function installed as an asynchronous interrupt handler. The capabilities of the asynchronous interrupt handler are limited. It cannot perform time consuming tasks and it cannot call into the User Interface Library. The function that is posted by `PostDeferredCall` contains code that cannot be executed at interrupt time. This feature is useful when external devices generate events during source program execution.

**QueueUserEvent**

Use `QueueUserEvent` to place a programmer-defined event in the `GetUserEvent` queue. Event numbers 1000 to 10000 are reserved for programmer-defined events. `GetUserEvent` receives programmer-defined events.

**FakeKeystroke**

Use `FakeKeystroke` to simulate a keystroke. This function has the same effect as actually pressing the key at the keyboard.

**Note:** The order in which the fake keystroke is received in relation to other events is not defined, so you must use this function with care.

**QuitUserInterface**

`QuitUserInterface` terminates the `RunUserInterface` function.
Chapter 4
User Interface Library Reference

This chapter describes the functions in the LabWindows User Interface Library. The User Interface Library Overview section contains general information about the User Interface Library functions and panels. The User Interface Library Function Reference section contains an alphabetical list of function descriptions.

User Interface Library Overview

This section contains general information about the User Interface Library functions and panels.

User Interface Function Panels

The User Interface Library function panels are grouped in a tree structure according to the types of operations performed. The User Interface Library function tree is shown in Table 4-1.

The bold headings in the tree are the names of function classes and subclasses. Function classes and subclasses are groups of related function panels. The headings in plain text are the names of individual function panels. Each User Interface Library function panel generates one function call. The names of the corresponding function calls are in bold italics to the right of the function panel names.

The function classes in the tree are described here.

- **Panels** is a class of functions that load/create, modify, and unload/discard user-defined panels.
- **Menu Structures** is a class of functions that load/create, modify, and unload/discard user-defined menu structures.
- **Controls/Graphs/Strip Charts** is a class of functions that create, control, modify, and discard controls and graphs.
- **Pop-up Panels** is a class of functions that install and interact with user-defined and predefined dialog boxes.
- **Callback Functions** is a class of functions that install user-defined callback functions that respond to user interface events and Windows messages.
• **User Interface Management** is a class of functions that control user input and screen operations.

• **Printing** is a class of functions that configure and generate hard copy output.

• **Miscellaneous** is a class of functions that do not fit into the other classes.

• **LW DOS Compatibility Functions** is a class of functions that are maintained for backwards compatibility with existing LabWindows for DOS applications.

Table 4-1. The User Interface Library Function Tree

<table>
<thead>
<tr>
<th>User Interface Library</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panels</td>
<td></td>
</tr>
<tr>
<td>Load Panel</td>
<td>LoadPanel</td>
</tr>
<tr>
<td>Load Panel (Extended)</td>
<td>LoadPanelEx</td>
</tr>
<tr>
<td>New Panel NewPanel</td>
<td></td>
</tr>
<tr>
<td>Discard Panel</td>
<td>DiscardPanel</td>
</tr>
<tr>
<td>Duplicate Panel</td>
<td>DuplicatePanel</td>
</tr>
<tr>
<td>Display Panel</td>
<td>DisplayPanel</td>
</tr>
<tr>
<td>Hide Panel</td>
<td>HidePanel</td>
</tr>
<tr>
<td>Get Active Panel</td>
<td>GetActivePanel</td>
</tr>
<tr>
<td>Set Active Panel</td>
<td>SetActivePanel</td>
</tr>
<tr>
<td>Validate Panel</td>
<td>ValidatePanel</td>
</tr>
<tr>
<td>Default Panel</td>
<td>DefaultPanel</td>
</tr>
<tr>
<td>Save Panel State</td>
<td>SavePanelState</td>
</tr>
<tr>
<td>Recall Panel State</td>
<td>RecallPanelState</td>
</tr>
<tr>
<td>Get Panel Attribute</td>
<td>GetPanelAttribute</td>
</tr>
<tr>
<td>Set Panel Attribute</td>
<td>SetPanelAttribute</td>
</tr>
<tr>
<td>Set Panel Position</td>
<td>SetPanelPos</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Menu Structures</th>
<th>Menu Bars</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Load Menu Bar</td>
<td>LoadMenuBar</td>
</tr>
<tr>
<td></td>
<td>Load Menu Bar (Extended)</td>
<td>LoadMenuBarEx</td>
</tr>
<tr>
<td></td>
<td>New Menu Bar</td>
<td>NewMenuBar</td>
</tr>
<tr>
<td></td>
<td>Discard Menu Bar</td>
<td>DiscardMenuBar</td>
</tr>
<tr>
<td></td>
<td>Set Panel Menu Bar</td>
<td>SetPanelMenuBar</td>
</tr>
<tr>
<td></td>
<td>Get Panel Menu Bar</td>
<td>GetPanelMenuBar</td>
</tr>
<tr>
<td></td>
<td>Get Menu Bar Attribute</td>
<td>GetMenuBarAttribute</td>
</tr>
<tr>
<td></td>
<td>Set Menu Bar Attribute</td>
<td>SetMenuBarAttribute</td>
</tr>
<tr>
<td></td>
<td>Empty Menu Bar</td>
<td>EmptyMenuBar</td>
</tr>
<tr>
<td></td>
<td>Get Shared Menu Bar Event Panel</td>
<td>GetSharedMenuBarEventPanel</td>
</tr>
</tbody>
</table>

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

<table>
<thead>
<tr>
<th>Menus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>New Menu</td>
<td>NewMenu</td>
</tr>
<tr>
<td>Discard Menu</td>
<td>DiscardMenu</td>
</tr>
<tr>
<td>Empty Menu</td>
<td>EmptyMenu</td>
</tr>
<tr>
<td>New SubMenu</td>
<td>NewSubMenu</td>
</tr>
<tr>
<td>Discard SubMenu</td>
<td>DiscardSubMenu</td>
</tr>
<tr>
<td>Run Popup Menu</td>
<td>RunPopupMenu</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Menu Items</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>New Menu Item</td>
<td>NewMenuItem</td>
</tr>
<tr>
<td>Discard Menu Item</td>
<td>DiscardMenuItem</td>
</tr>
<tr>
<td>Insert Separator</td>
<td>InsertSeparator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controls/Graphs/Strip Charts</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>General Functions</td>
<td></td>
</tr>
<tr>
<td>New Control</td>
<td>NewCtrl</td>
</tr>
<tr>
<td>Duplicate Control</td>
<td>DuplicateCtrl</td>
</tr>
<tr>
<td>Discard Control</td>
<td>DiscardCtrl</td>
</tr>
<tr>
<td>Get Active Control</td>
<td>GetActiveCtrl</td>
</tr>
<tr>
<td>Set Active Control</td>
<td>SetActiveCtrl</td>
</tr>
<tr>
<td>Default Control Value</td>
<td>DefaultCtrl</td>
</tr>
<tr>
<td>Get Control Value</td>
<td>GetCtrlVal</td>
</tr>
<tr>
<td>Set Control Value</td>
<td>SetCtrlVal</td>
</tr>
<tr>
<td>Get Control Attribute</td>
<td>GetCtrlAttribute</td>
</tr>
<tr>
<td>Set Control Attribute</td>
<td>SetCtrlAttribute</td>
</tr>
<tr>
<td>Get Control Bounding Rectangle</td>
<td>GetCtrlBoundingRect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>List (Label/Value) Controls</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert List Item</td>
<td>InsertListItem</td>
</tr>
<tr>
<td>Replace List Item</td>
<td>ReplaceListItem</td>
</tr>
<tr>
<td>Delete List Item</td>
<td>DeleteListItem</td>
</tr>
<tr>
<td>Get Value From Index</td>
<td>GetValueFromIndex</td>
</tr>
<tr>
<td>Get Value Length From Index</td>
<td>GetValueLengthFromIndex</td>
</tr>
<tr>
<td>Get Index From Value</td>
<td>GetIndexFromValue</td>
</tr>
<tr>
<td>Get Control Index</td>
<td>GetCtrlIndex</td>
</tr>
<tr>
<td>Set Control Index</td>
<td>SetCtrlIndex</td>
</tr>
<tr>
<td>Clear List Control</td>
<td>ClearListCtrl</td>
</tr>
<tr>
<td>Get Number of List Items</td>
<td>GetNumListItems</td>
</tr>
<tr>
<td>Get List Item Image</td>
<td>GetListItemImage</td>
</tr>
<tr>
<td>Set List Item Image</td>
<td>SetListItemImage</td>
</tr>
<tr>
<td>Get Label From Index</td>
<td>GetLabelFromIndex</td>
</tr>
<tr>
<td>Get Label Length From Index</td>
<td>GetLabelLengthFromIndex</td>
</tr>
<tr>
<td>Is List Item Checked</td>
<td>IsListItemChecked</td>
</tr>
<tr>
<td>Check List Item</td>
<td>CheckListItem</td>
</tr>
<tr>
<td>Get Number of Checked Items</td>
<td>GetNumCheckedItems</td>
</tr>
</tbody>
</table>

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

<table>
<thead>
<tr>
<th>Text Boxes</th>
<th>InsertTextBoxLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Text Box Line</td>
<td>ReplaceTextBoxLine</td>
</tr>
<tr>
<td>Replace Text Box Line</td>
<td>DeleteTextBoxLine</td>
</tr>
<tr>
<td>Delete Text Box Line</td>
<td>GetNumTextBoxLines</td>
</tr>
<tr>
<td>Get Number of Text Box Lines</td>
<td>ResetTextBox</td>
</tr>
<tr>
<td>Reset Text Box</td>
<td>GetTextBoxLine</td>
</tr>
<tr>
<td>Get Text Box Line</td>
<td>GetTextBoxLineLength</td>
</tr>
<tr>
<td>Get Text Box Line Length</td>
<td></td>
</tr>
</tbody>
</table>

Graphs and Strip Charts

Graph Plotting and Deleting

<table>
<thead>
<tr>
<th>Plot X</th>
<th>PlotX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot Y</td>
<td>PlotY</td>
</tr>
<tr>
<td>Plot X-Y</td>
<td>PlotXY</td>
</tr>
<tr>
<td>Plot Waveform</td>
<td>PlotWaveform</td>
</tr>
<tr>
<td>Plot Point</td>
<td>PlotPoint</td>
</tr>
<tr>
<td>Plot Text</td>
<td>PlotText</td>
</tr>
<tr>
<td>Plot Line</td>
<td>PlotLine</td>
</tr>
<tr>
<td>Plot Rectangle</td>
<td>PlotRectangle</td>
</tr>
<tr>
<td>Plot Polygon</td>
<td>PlotPolygon</td>
</tr>
<tr>
<td>Plot Oval</td>
<td>PlotOval</td>
</tr>
<tr>
<td>Plot Arc</td>
<td>PlotArc</td>
</tr>
<tr>
<td>Plot Intensity</td>
<td>PlotIntensity</td>
</tr>
<tr>
<td>Plot Scaled Intensity</td>
<td>PlotScaledIntensity</td>
</tr>
<tr>
<td>Plot Bitmap</td>
<td>PlotBitmap</td>
</tr>
<tr>
<td>Delete Graph Plot</td>
<td>DeleteGraphPlot</td>
</tr>
<tr>
<td>Get Plot Attribute</td>
<td>GetPlotAttribute</td>
</tr>
<tr>
<td>Set Plot Attribute</td>
<td>SetPlotAttribute</td>
</tr>
<tr>
<td>Refresh Graph</td>
<td>RefreshGraph</td>
</tr>
</tbody>
</table>

Graph Cursors

<table>
<thead>
<tr>
<th>Get Graph Cursor</th>
<th>GetGraphCursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Graph Cursor</td>
<td>SetGraphCursor</td>
</tr>
<tr>
<td>Get Active Graph Cursor</td>
<td>GetActiveGraphCursor</td>
</tr>
<tr>
<td>Set Active Graph Cursor</td>
<td>SetActiveGraphCursor</td>
</tr>
<tr>
<td>Get Graph Cursor Index</td>
<td>GetGraphCursorIndex</td>
</tr>
<tr>
<td>Set Graph Cursor Index</td>
<td>SetGraphCursorIndex</td>
</tr>
<tr>
<td>Get Cursor Attribute</td>
<td>GetCursorAttribute</td>
</tr>
<tr>
<td>Set Cursor Attribute</td>
<td>SetCursorAttribute</td>
</tr>
</tbody>
</table>

Strip Chart Traces

<table>
<thead>
<tr>
<th>Plot Strip Chart</th>
<th>PlotStripChart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot Strip Chart Point</td>
<td>PlotStripChartPoint</td>
</tr>
</tbody>
</table>

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

| Function                                      | Description                          
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Strip Chart</td>
<td>ClearStripChart</td>
</tr>
<tr>
<td>Get Trace Attribute</td>
<td>GetTraceAttribute</td>
</tr>
<tr>
<td>Set Trace Attribute</td>
<td>SetTraceAttribute</td>
</tr>
<tr>
<td><strong>Axis Scaling</strong></td>
<td></td>
</tr>
<tr>
<td>Get Axis Scaling Mode</td>
<td>GetAxisScalingMode</td>
</tr>
<tr>
<td>Set Axis Scaling Mode</td>
<td>SetAxisScalingMode</td>
</tr>
<tr>
<td>Get Axis Range (Obsolete)</td>
<td>GetAxisRange</td>
</tr>
<tr>
<td>Set Axis Range (Obsolete)</td>
<td>SetAxisRange</td>
</tr>
<tr>
<td><strong>Axis Label Strings</strong></td>
<td></td>
</tr>
<tr>
<td>Insert Axis Item</td>
<td>InsertAxisItem</td>
</tr>
<tr>
<td>Replace Axis Item</td>
<td>ReplaceAxisItem</td>
</tr>
<tr>
<td>Delete Axis Item</td>
<td>DeleteAxisItem</td>
</tr>
<tr>
<td>Clear Axis Items</td>
<td>ClearAxisItems</td>
</tr>
<tr>
<td>Get Number of Axis Items</td>
<td>GetNumAxisItems</td>
</tr>
<tr>
<td>Get Axis Item Label and Value</td>
<td>GetAxisItem</td>
</tr>
<tr>
<td>Get Axis Item Label Length</td>
<td>GetAxisItemLabelLength</td>
</tr>
<tr>
<td><strong>Pictures</strong></td>
<td></td>
</tr>
<tr>
<td>Display Image File</td>
<td>DisplayImageFile</td>
</tr>
<tr>
<td>Delete Image</td>
<td>DeleteImage</td>
</tr>
<tr>
<td>Get Image Info (Obsolete)</td>
<td>GetImageInfo</td>
</tr>
<tr>
<td>Get Image Bits (Obsolete)</td>
<td>GetImageBits</td>
</tr>
<tr>
<td>Set Image Bits (Obsolete)</td>
<td>SetImageBits</td>
</tr>
<tr>
<td>Alloc Image Bits (Obsolete)</td>
<td>AllocImageBits</td>
</tr>
<tr>
<td><strong>Canvas</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Drawing</strong></td>
<td></td>
</tr>
<tr>
<td>Draw Point</td>
<td>CanvasDrawPoint</td>
</tr>
<tr>
<td>Draw Line</td>
<td>CanvasDrawLine</td>
</tr>
<tr>
<td>Draw Line To</td>
<td>CanvasDrawLineTo</td>
</tr>
<tr>
<td>Draw Rectangle</td>
<td>CanvasDrawRect</td>
</tr>
<tr>
<td>Dim Rectangle</td>
<td>CanvasDimRect</td>
</tr>
<tr>
<td>Draw Rounded Rectangle</td>
<td>CanvasDrawRoundedRect</td>
</tr>
<tr>
<td>Draw Oval</td>
<td>CanvasDrawOval</td>
</tr>
<tr>
<td>Draw Arc</td>
<td>CanvasDrawArc</td>
</tr>
<tr>
<td>Draw Poly</td>
<td>CanvasDrawPoly</td>
</tr>
<tr>
<td>Draw Text in Rectangle</td>
<td>CanvasDrawText</td>
</tr>
<tr>
<td>Draw Text at Point</td>
<td>CanvasDrawTextAtPoint</td>
</tr>
<tr>
<td>Draw Bitmap</td>
<td>CanvasDrawBitmap</td>
</tr>
<tr>
<td>Scroll</td>
<td>CanvasScroll</td>
</tr>
<tr>
<td>Invert Rectangle</td>
<td>CanvasInvertRect</td>
</tr>
<tr>
<td>Clear</td>
<td>CanvasClear</td>
</tr>
</tbody>
</table>

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

<table>
<thead>
<tr>
<th>Batch Drawing</th>
<th>CanvasStartBatchDraw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Batch Drawing</td>
<td>CanvasStartBatchDraw</td>
</tr>
<tr>
<td>End Batch Drawing</td>
<td>CanvasEndBatchDraw</td>
</tr>
<tr>
<td>Pens</td>
<td></td>
</tr>
<tr>
<td>Set Pen Position</td>
<td>CanvasSetPenPosition</td>
</tr>
<tr>
<td>Get Pen Position</td>
<td>CanvasGetPenPosition</td>
</tr>
<tr>
<td>Set Pen Attributes To Defaults</td>
<td>CanvasDefaultPen</td>
</tr>
<tr>
<td>Clipping</td>
<td></td>
</tr>
<tr>
<td>Set Clipping Rectangle</td>
<td>CanvasSetClipRect</td>
</tr>
<tr>
<td>Get Clipping Rectangle</td>
<td>CanvasGetClipRect</td>
</tr>
<tr>
<td>Accessing Pixel Values</td>
<td></td>
</tr>
<tr>
<td>Get a Single Pixel Value</td>
<td>CanvasGetPixel</td>
</tr>
<tr>
<td>Get Pixel Values</td>
<td>CanvasGetPixels</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
</tr>
<tr>
<td>Update Canvas</td>
<td>CanvasUpdate</td>
</tr>
<tr>
<td>Timers</td>
<td></td>
</tr>
<tr>
<td>Reset Timer</td>
<td>ResetTimer</td>
</tr>
<tr>
<td>Suspend Timer Callbacks</td>
<td>SuspendTimerCallbacks</td>
</tr>
<tr>
<td>Resume Timer Callbacks</td>
<td>ResumeTimerCallbacks</td>
</tr>
<tr>
<td>Pop-up Panels</td>
<td></td>
</tr>
<tr>
<td>Install Popup</td>
<td>InstallPopup</td>
</tr>
<tr>
<td>Remove Popup</td>
<td>RemovePopup</td>
</tr>
<tr>
<td>Message Popup</td>
<td>MessagePopup</td>
</tr>
<tr>
<td>Confirm Popup</td>
<td>ConfirmPopup</td>
</tr>
<tr>
<td>Prompt Popup</td>
<td>PromptPopup</td>
</tr>
<tr>
<td>Generic Message</td>
<td>GenericMessagePopup</td>
</tr>
<tr>
<td>File Select Popup</td>
<td>FileSelectPopup</td>
</tr>
<tr>
<td>Multifile Select Popup</td>
<td>MultiFileSelectPopup</td>
</tr>
<tr>
<td>Directory Select Popup</td>
<td>DirSelectPopup</td>
</tr>
<tr>
<td>X Graph Popup</td>
<td>XGraphPopup</td>
</tr>
<tr>
<td>Y Graph Popup</td>
<td>YGraphPopup</td>
</tr>
<tr>
<td>X-Y Graph Popup</td>
<td>XYGraphPopup</td>
</tr>
<tr>
<td>Waveform Graph Popup</td>
<td>WaveformGraphPopup</td>
</tr>
<tr>
<td>Get System Popups Attribute</td>
<td>GetSystemPopupsAttribute</td>
</tr>
<tr>
<td>Set System Popups Attribute</td>
<td>SetSystemPopupsAttribute</td>
</tr>
<tr>
<td>Font Select Popup</td>
<td>FontSelectPopup</td>
</tr>
<tr>
<td>Set Font Select Popup Defaults</td>
<td>SetFontPopupDefaults</td>
</tr>
<tr>
<td>Callback Functions</td>
<td></td>
</tr>
<tr>
<td>Install Main Callback</td>
<td>InstallMainCallback</td>
</tr>
<tr>
<td>Install Control Callback</td>
<td>InstallCtrlCallback</td>
</tr>
<tr>
<td>Install Panel Callback</td>
<td>InstallPanelCallback</td>
</tr>
<tr>
<td>Install Menu Callback</td>
<td>InstallMenuCallback</td>
</tr>
</tbody>
</table>

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

| Install Menu Dimmer Callback | InstallMenuDimmerCallback |
| Post Deferred Call | PostDeferredCall |
| **Windows Interrupt Support** |  |
| Register Windows Msg Callback | RegisterWinMsgCallback |
| Unregister Windows Msg Callback | UnRegisterWinMsgCallback |
| Get CVI Window Handle | GetCVIWindowHandle |
| Get CVI Task Handle | GetCVITaskHandle |

**User Interface Management**
- Run User Interface
- Quit User Interface
- Get User Event
- Set Input Mode
- Process Draw Events
- Process System Events
- Queue User Event
- Set Idle Event Rate
- Fake Keystroke
- Get Sleep Policy
- Set Sleep Policy

**Printing**
- Get Print Attribute
- Set Print Attribute
- Print Control
- Print Panel
- Print Text File
- Print Text Buffer

**Mouse and Cursor**
- Get Wait Cursor
- Set Wait Cursor
- Get Mouse Cursor
- Set Mouse Cursor
- Get Global Mouse State
- Get Relative Mouse State

**Rectangles and Points**

**Creating and Modifying**
- Make Rect
- Set Rect Coordinates
- Set Rect Coords From Points
- Set Bottom Edge of Rect
- Set Right Edge of Rect
- Set Center Point of Rect

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

<table>
<thead>
<tr>
<th>Offset Rect</th>
<th>RectOffset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move Rect</td>
<td>RectMove</td>
</tr>
<tr>
<td>Grow (or Shrink) Rect</td>
<td>RectGrow</td>
</tr>
<tr>
<td>Make Point</td>
<td>MakePoint</td>
</tr>
<tr>
<td>Set Point Coordinates</td>
<td>PointSet</td>
</tr>
</tbody>
</table>

**Retrieving and Comparing Values**

<table>
<thead>
<tr>
<th>Get Rect Bottom</th>
<th>RectBottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Rect Right</td>
<td>RectRight</td>
</tr>
<tr>
<td>Get Rect Center</td>
<td>RectCenter</td>
</tr>
<tr>
<td>Are Rects Equal?</td>
<td>RectEqual</td>
</tr>
<tr>
<td>Is Rect Empty?</td>
<td>RectEmpty</td>
</tr>
<tr>
<td>Does Rect Contain Point?</td>
<td>RectContainsPoint</td>
</tr>
<tr>
<td>Does Rect Contain Rect?</td>
<td>RectContainsRect</td>
</tr>
<tr>
<td>Are Rects the Same Size?</td>
<td>RectSameSize</td>
</tr>
<tr>
<td>Calculate Rect Union</td>
<td>RectUnion</td>
</tr>
<tr>
<td>Calculate Rect Intersection</td>
<td>RectIntersection</td>
</tr>
<tr>
<td>Are Points Equal?</td>
<td>PointEqual</td>
</tr>
<tr>
<td>Calculate Point Pinned to Rect</td>
<td>PointPinnedToRect</td>
</tr>
</tbody>
</table>

**Bitmaps**

<table>
<thead>
<tr>
<th>Create New Bitmap</th>
<th>NewBitmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Bitmap From a File</td>
<td>GetBitmapFromFile</td>
</tr>
<tr>
<td>Get Bitmap From a Control</td>
<td>GetCtrlBitmap</td>
</tr>
<tr>
<td>Get Control Display Bitmap</td>
<td>GetCtrlDisplayBitmap</td>
</tr>
<tr>
<td>Get Panel Display Bitmap</td>
<td>GetPanelDisplayBitmap</td>
</tr>
<tr>
<td>Get Bitmap Info</td>
<td>GetBitmapInfo</td>
</tr>
<tr>
<td>Get Bitmap Data</td>
<td>GetBitmapData</td>
</tr>
<tr>
<td>Alloc Bitmap Data</td>
<td>AllocBitmapData</td>
</tr>
<tr>
<td>Set Control Bitmap</td>
<td>SetCtrlBitmap</td>
</tr>
<tr>
<td>Discard Bitmap</td>
<td>DiscardBitmap</td>
</tr>
</tbody>
</table>

**Clipboard**

<table>
<thead>
<tr>
<th>Get Text From Clipboard</th>
<th>ClipboardGetText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put Text On Clipboard</td>
<td>ClipboardPutText</td>
</tr>
<tr>
<td>Get Bitmap From Clipboard</td>
<td>ClipboardGetBitmap</td>
</tr>
<tr>
<td>Put Bitmap on Clipboard</td>
<td>ClipboardPutBitmap</td>
</tr>
</tbody>
</table>

**Miscellaneous**

<table>
<thead>
<tr>
<th>Make Color</th>
<th>MakeColor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get 3d Border Colors</td>
<td>Get3dBorderColors</td>
</tr>
<tr>
<td>Create Meta Font</td>
<td>CreateMetaFont</td>
</tr>
<tr>
<td>Get Text Display Size</td>
<td>GetTextDisplaySize</td>
</tr>
<tr>
<td>Get Screen Size</td>
<td>GetScreenSize</td>
</tr>
<tr>
<td>Get System Attribute</td>
<td>GetSystemAttribute</td>
</tr>
<tr>
<td>Set System Attribute</td>
<td>SetSystemAttribute</td>
</tr>
</tbody>
</table>

(continues)
Table 4-1. The User Interface Library Function Tree (Continued)

<table>
<thead>
<tr>
<th>LW DOS Compatibility Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure Printer</td>
<td>ConfigurePrinter</td>
</tr>
<tr>
<td>Display PCX File</td>
<td>DisplayPCXFile</td>
</tr>
<tr>
<td>DOS Color to RGB</td>
<td>DOSColorToRGB</td>
</tr>
<tr>
<td>DOS Compatibility Window</td>
<td>DOSCompatWindow</td>
</tr>
<tr>
<td>Get Error String</td>
<td>GetUILLErrorString</td>
</tr>
</tbody>
</table>

**Hard Copy Output**

**Compatible Printers**

Under Windows, LabWindows/CVI can print to any graphics printer with a Windows compatible driver.

Under UNIX, LabWindows/CVI can print to any graphics printer compatible with the `xpr` printing utility under the X Window System.

In addition, output can be redirected to a disk file.

**Obtaining Hard Copy Output**

You can use the following functions to obtain hard copy output.

- PrintPanel
- PrintCtrl
- PrintTextFile
- PrintTextBuffer

In addition, the following functions can also produce hard copy output if you select the hard copy option from the following pop-ups.

- XGraphPopup
- XYGraphPopup
- YGraphPopup
- WaveformGraphPopup

**Configuring Your System for Hard Copy Output**

Under Windows, the printer driver and output port are set using the Printers utility in the Windows Control Panel. Under UNIX, the default printer is set using the `PRINTER` environment variable or the printcap file and the printer filter is set in the `.Xdefaults` file. Hard copy attributes can be set interactively through a dialog box or programmatically through the `SetPrinterAttribute` function. These attributes are discussed in the section, *Generating Hard Copy Output*, in Chapter 3, *Programming with the User Interface Library*. 
**RGB Color Values**

The following functions can use RGB color values as input parameters.

- PlotArc
- PlotLine
- PlotOval
- PlotPoint
- PlotPolygon
- PlotRectangle
- PlotText
- PlotWaveform
- PlotX
- PlotXY
- PlotY
- PlotIntensity
- SetImageBits
- SetCtrlAttribute
- SetCursorAttribute
- SetMenuBarAttribute
- SetPanelAttribute
- SetTraceAttribute
- SetPlotAttribute

An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. Common colors are shown in Table 3-3 of Chapter 3, *Programming with the User Interface Library*.

You can also use the User Interface Library function, `MakeColor`, to create an RGB value from red, green, and blue color components.

**Fonts**

The following functions use font values as input parameters.

- PlotText
- SetPanelAttribute
- SetCtrlAttribute
- FontSelectPopup
- SetFontSelectPopupDefaults

See the section *Using Fonts* in Chapter 1, *User Interface Concepts*, for more information on font values.
Shortcut Keys

The following functions can use shortcut key values as input parameters.

- SetMenuBarAttribute
- SetCtrlAttribute

In your code, shortcut keys are 4-byte integers consisting of three bit fields, 0x00MMVVAA, where:

- **MM** = the modifier key
- **VV** = the virtual key
- **AA** = the ASCII key

When constructing a shortcut key, modifier keys are bit-wise OR’ed with a virtual key or an ASCII key. Either the ASCII field or the virtual key field will be all zeros. For example:

```
VAL_SHIFT_MODIFIER | VAL_F1_VKEY
```

produces a shortcut key of SHIFT-F1, and

```
VAL_MENUKEY_MODIFIER | 'D'
```

produces a shortcut key of <Ctrl-D>. Virtual keys do not require modifiers but ASCII keys require at least one modifier.

Table 3-7 in Chapter 3, *Programming with the User Interface Library* shows the modifiers and virtual keys for shortcut keys.

Plot Array Data Types

The following graphing functions have array parameters whose data values are interpreted based on other parameters called *data type specifiers*.

- PlotPolygon
- PlotStripChart
- PlotWaveform
- PlotX
- PlotXY
- PlotY
- PlotIntensity
- WaveformGraphPopup
- XGraphPopup
- XYGraphPopup
- YGraphPopup

A data type specifier determines the data type of an array and must be one of the values shown in the following list.
VAL_CHAR
VAL_INTEGER
VAL_SHORT_INTEGER
VAL_FLOAT
VAL_DOUBLE
VAL_UNSIGNED_SHORT_INTEGER
VAL_UNSIGNED_INTEGER
VAL_UNSIGNED_CHAR

Include Files

The User Interface Library provides an include file that contains function declarations and defined constants for all of the library routines. The include file is named userint.h. You must include this file in all code modules that reference the User Interface Library.

Reporting Errors

All the functions in the User Interface Library return an integer code containing the result of the call. If the return code is negative, an error occurred. Otherwise, the function completed successfully. Refer to Appendix A, Error Conditions, for a complete list of error codes. You can also use the GetUILErrorString function to convert the error number returned into a text error message.

User Interface Library Function Reference

This section describes each function in the User Interface Library. The functions are in alphabetical order.

AllocBitmapData

int status = AllocBitmapData(int bitmapID, int **colorTable, char **bits,
                                      unsigned char **mask);

Purpose

Allocates the buffers necessary for calling GetBitmapData on a bitmap. If you use GetBitmapInfo, you must allocate the buffers yourself.

You must free the buffers when you are done with them.
## Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>bitmapID</th>
<th>integer</th>
<th>The ID of the bitmap object containing the image. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, or GetPanelDisplayBitmap.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>colorTable</th>
<th>pointer to integer</th>
<th>A pointer variable into which the address of the allocated color table buffer is placed.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bits</td>
<td>pointer to unsigned char</td>
<td>A pointer variable into which the address of the allocated bits data buffer is placed.</td>
</tr>
<tr>
<td></td>
<td>mask</td>
<td>pointer to unsigned char</td>
<td>A pointer variable into which the address of the allocated mask buffer is placed.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

### Parameter Discussion

You may pass NULL for any of the `colorTable`, `bits`, or `mask` parameters if you do not want the corresponding buffer to be allocated.

If the image does not exist, the `colorTable`, `bits`, and `mask` parameters are set to NULL. The `colorTable` parameter is set to NULL if the pixel depth of the image is greater than 8. The `mask` parameter is set to NULL if the image does not have a mask.

**Warning:** You must free the colorTable, bitmap, and mask buffers when you are done with them. Use the ANSI C Library `free` function.

### See Also

GetBitmapData, GetBitmapInfo
## AllocImageBits

```c
int status = AllocImageBits(int panelHandle, int controlID, int imageSize,
    int **colorTable, unsigned char **bitmap,
    unsigned char **mask)
```

### Purpose

Allocates the buffers necessary for calling `GetImageBits` on an existing image. This function provides an alternative to calling `GetImageInfo` and allocating the buffers yourself.

The following control types can contain images:

- picture controls
- picture rings
- picture buttons
- graph controls

You must free the buffers allocated by this function when you are done with them.

### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><code>panelHandle</code></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>controlID</code></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the user Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td><code>imageID</code></td>
<td>integer</td>
<td>For a picture ring, the zero-based index of an image in the ring. For a graph, the <code>plotHandle</code> returned from <code>PlotBitmap</code>. For picture controls and buttons, this is ignored.</td>
</tr>
<tr>
<td>Output</td>
<td><code>colorTable</code></td>
<td>pointer to integer</td>
<td>A pointer variable into which the address of the allocated color table buffer is placed.</td>
</tr>
<tr>
<td></td>
<td><code>bitmap</code></td>
<td>pointer to unsigned char</td>
<td>A pointer variable into which the address of the allocated bitmap buffer is placed.</td>
</tr>
<tr>
<td></td>
<td><code>mask</code></td>
<td>pointer to unsigned char</td>
<td>A pointer variable into which the address of the allocated mask buffer is placed.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>

Parameter Discussion

You may pass NULL for any of the colorTable, bitmap, and mask parameters if you do not want the corresponding buffer to be allocated.

If the image does not exist, the colorTable, bitmap, and mask parameters are set to NULL. The mask parameter is set to NULL if the image does not have a mask.

Warning: You must free the colorTable, bitmap, and mask buffers when you are done with them. Use the ANSI C Library free function.

You can now use a 24-bit pixel depth in the four picture control image bits functions. When you do, the color table parameters to the functions are not used. The array of bits contains RGB values rather than indexes into the color table. Each RGB value in the array of bits represented by a 3-byte value of the form 0xRRGGBB

where RR, GG, and BB represent the red, green and blue intensity of the color. The RR byte should always be at the lowest memory address of the three bytes.

See Also

GetImageBits, SetImageBits

CanvasClear

int status = CanvasClear (int panelHandle, int controlID, Rect rect);

Purpose

Restores the specified rectangular area of a canvas control to the background color of the canvas control. The background color of the canvas control is determined by the ATTR_PICT_BGCOLOR attribute.

This operation is not restricted to the canvas clipping rectangle.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>rect</td>
<td>Rect</td>
<td></td>
<td>A Rect structure specifying the location and size of the rectangle to be cleared. Use VAL_ENTIRE_OBJECT to specify the entire canvas.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect

CanvasDefaultPen

```c
int status = CanvasDefaultPen(int panelHandle, int controlID);
```

Purpose

Sets all of the attributes of the canvas pen to the default values. The defaults are shown in the following table.

<table>
<thead>
<tr>
<th>Canvas Pen Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_PEN_WIDTH</td>
<td>1</td>
</tr>
<tr>
<td>ATTR_PEN_STYLE</td>
<td>VAL_SOLID</td>
</tr>
<tr>
<td>ATTR_PEN_COLOR</td>
<td>VAL_BLACK</td>
</tr>
<tr>
<td>ATTR_PEN_FILL_COLOR</td>
<td>VAL_BLACK</td>
</tr>
<tr>
<td>ATTR_PEN_MODE</td>
<td>VAL_COPY_MODE</td>
</tr>
<tr>
<td>ATTR_PEN_PATTERN</td>
<td>A solid pattern, expressed as an array of 8 unsigned characters, each of which is 0xFF.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

CanvasSetPenPosition, CanvasGetPenPosition, CanvasDrawLineTo

CanvasDimRect

```c
int status = CanvasDimRect(int panelHandle, int controlID, Rect rect);
```

Purpose

Overlays a checkerboard pattern in the specified rectangular area of a canvas control. This has the visual effect of dimming objects within the area.

The checkerboard pattern is drawn using current values of the following attribute.

ATTR_PEN_FILL_COLOR
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A Rect structure specifying the location and size of the area to be dimmed. Use VAL_ENTIRE_OBJECT to specify the entire canvas.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect

CanvasDrawArc

```c
int status = CanvasDrawArc(int panelHandle, int controlID, Rect rect,
int drawMode, int beginningAngle, int arcAngle);
```

Purpose

Draws an arc on the canvas control. The arc is defined by specifying a rectangle that encloses the arc, along with a beginning angle (in tenths of degrees) and an arc angle (in tenths of degrees).

The arc is a section of an oval. A beginning angle of 0 indicates that the arc starts at the midpoint of the right edge of the rectangle. The arc angle indicates how far around the oval (counterclockwise, up to 3600) the arc is drawn.

The frame of the arc is drawn using the current value of the following attributes:

- ATTR_PEN_COLOR
- ATTR_PEN_MODE
- ATTR_PEN_WIDTH
- ATTR_PEN_STYLE (ignored in Windows when pen width is greater than 1)
The interior of the arc is drawn using the current value of the following attributes.

ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN

The frame of the arc does not include the radius lines going from the center of the oval to the end points of the arc.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A Rect structure specifying the location and size of the rectangle within which to draw the arc.</td>
</tr>
<tr>
<td></td>
<td>drawMode</td>
<td>integer</td>
<td>Specifies whether the arc's frame or interior (or both) are drawn. Valid values: VAL_DRAW_FRAME, VAL_DRAW_INTERIOR, VAL_DRAW_FRAME_AND_INTERIOR</td>
</tr>
<tr>
<td></td>
<td>beginningAngle</td>
<td>integer</td>
<td>The starting angle of the arc, in tenths of degrees. 0 indicates the arc starts at the midpoint of the right edge of the rectangle. 900 indicates that the arc starts at the midpoint of the top edge of the rectangle. Negative values are valid.</td>
</tr>
<tr>
<td></td>
<td>arcAngle</td>
<td>integer</td>
<td>How far around the oval (counter-clockwise, up to 3600) the arc is drawn. Specified in tenths of degrees. Negative values are valid.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect, CanvasDrawOval
CanvasDrawBitmap

```c
int status = CanvasDrawBitmap(int panelHandle, int controlID, int bitmapID, Rect sourceRect, Rect destinationRect);
```

### Purpose

Draws a bitmap image (or portion thereof) in the specified destination rectangle on the canvas control.

### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>controlID</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>bitmapID</td>
<td>The ID of the bitmap object containing the image. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, or GetPanelDisplayBitmap.</td>
</tr>
<tr>
<td>sourceRect</td>
<td>A Rect structure specifying the portion of the bitmap to be drawn. The values are in terms of the pixel coordinates of the bitmap. The origin (0, 0) is at the upper left corner of the bitmap. Use VAL_ENTIRE_OBJECT to specify the entire image.</td>
</tr>
<tr>
<td>destinationRect</td>
<td>A Rect structure specifying the size and location of the area in which the bitmap image is to be drawn on the canvas control. If sourceRect and destinationRect are not the same size, the bitmap is stretched or shrunk to fit.</td>
</tr>
</tbody>
</table>

### Return Value

| status | integer | Refer to Appendix A for error codes. |
Parameter Discussion

If you want the destination rectangle to be same size as the source rectangle, you can set the **height** and **width** in **destinationRect** to **VAL_KEEP_SAME_SIZE**.

If you want the bitmap to stretch to fit the size of the canvas, pass **VAL_ENTIRE_OBJECT** as **destinationRect**.

Example

The following code copies a bitmap image, without any stretching or shrinking, to the canvas control, starting 20 pixels below the top edge of the canvas, and 30 pixels to the right of left edge of the canvas.

```c
CanvasDrawBitmap (panelHandle, controlID, bitmapID, VAL_ENTIRE_OBJECT, 
    MakeRect(20,30, VAL_KEEP_SAME_SIZE, VAL_KEEP_SAME_SIZE));
```

See also

MakeRect

---

**CanvasDrawLine**

```c
int  status = CanvasDrawLine (int panelHandle, int controlID, Point start, 
    Point end);
```

Purpose

Draws a line between two specified points.

The line is drawn using the current value of the following attributes.

- **ATTR_PEN_COLOR**
- **ATTR_PEN_MODE**
- **ATTR_PEN_WIDTH**
- **ATTR_PEN_STYLE**  (ignored in Windows when pen width is greater than 1)
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>start</td>
<td>Point</td>
<td>A Point structure specifying the location at which the line begins.</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td>Point</td>
<td>A Point structure specifying the location at which the line ends.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakePoint, CanvasDrawLineTo
CanvasDrawLineTo

```c
int status = CanvasDrawLineTo (int panelHandle, int controlID, Point end);
```

**Purpose**

Draws a line between the current pen position and a specified end point, and sets the pen position to the end point.

The line is drawn using the current value of the following attributes.

- `ATTR_PEN_COLOR`
- `ATTR_PEN_MODE`
- `ATTR_PEN_WIDTH`
- `ATTR_PEN_STYLE` (ignored in Windows when pen width is greater than 1)

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td>Point</td>
<td>A Point structure specifying the location at which the line ends.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

**See also**

`MakePoint`, `CanvasGetPenPosition`, `CanvasSetPenPosition`, `CanvasDefaultPen`, `CanvasDrawLine`, `CanvasDrawOval` 

CanvasDrawOval

```c
int status = CanvasDrawOval (int panelHandle, int controlID, Rect rect, int drawMode);
```

**Purpose**

Draws an oval on the canvas control within the specified rectangle.
The frame of the oval is drawn using the current value of the following attributes:

ATTR_PEN_COLOR  
ATTR_PEN_MODE  
ATTR_PEN_WIDTH  
ATTR_PEN_STYLE  (ignored in Windows when pen width is greater than 1)

The interior of the oval is drawn using the current value of the following attributes:

ATTR_PEN_FILL_COLOR  
ATTR_PEN_MODE  
ATTR_PEN_PATTERN

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A Rect structure specifying the location and size of the rectangle within which to draw the oval.</td>
</tr>
<tr>
<td></td>
<td>drawMode</td>
<td>integer</td>
<td>Specifies whether the oval’s frame or interior (or both) are drawn. Valid values: VAL_DRAW_FRAME, VAL_DRAW_INTERIOR, VAL_DRAW_FRAME_AND_INTERIOR</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect, CanvasDrawArc
CanvasDrawPoint

int status = CanvasDrawPoint(int panelHandle, int controlID, Point point);

Purpose

Draws a point on the canvas control as the specified position.

The point is drawn using the current value of the following attributes:

ATTR_PEN_COLOR
ATTR_PEN_MODE
ATTR_PEN_WIDTH

At pen widths of greater than 1, the point may appear to be non-circular.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>point</td>
<td>Point</td>
<td>A Point structure specifying the location at which to draw the point.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect, CanvasDrawArc
CanvasDrawPoly

```c
int status = CanvasDrawPoly(int panelHandle, int controlID, int numberOfPoints, Point points[], int wrap, int drawMode);
```

**Purpose**

Draws a polygon on the canvas control by connecting the specified points.

The frame of the polygon is drawn using the current value of the following attributes:

- `ATTR_PEN_COLOR`
- `ATTR_PEN_MODE`
- `ATTR_PEN_WIDTH`
- `ATTR_PEN_STYLE` (ignored in Windows when pen width is greater than 1)

The interior of the polygon is drawn using the current value of the following attributes:

- `ATTR_PEN_FILL_COLOR`
- `ATTR_PEN_MODE`
- `ATTR_PEN_PATTERN`

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>panelHandle</code></th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>controlID</code></td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td></td>
<td>The number of vertices in the polygon.</td>
</tr>
<tr>
<td>points</td>
<td>Point array</td>
<td></td>
<td>An array of Point structures specifying the locations of the vertices of the polygon.</td>
</tr>
<tr>
<td>wrap</td>
<td>integer</td>
<td></td>
<td>A nonzero value specifies that a line is drawn between last point and first point, thereby closing the polygon frame. This value is ignored when drawing only the interior.</td>
</tr>
<tr>
<td>drawMode</td>
<td>integer</td>
<td></td>
<td>Specifies whether the polygon’s frame or interior (or both) are drawn. Valid values:</td>
</tr>
</tbody>
</table>

- `VAL_DRAW_FRAME`
- `VAL_DRAW_INTERIOR`
- `VAL_DRAW_FRAME_AND_INTERIOR`
Return Value

| status | integer | Refer to Appendix A for error codes. |

CanvasDrawRect

```c
int status = CanvasDrawRect (int panelHandle, int controlID, Rect rect, int drawMode);
```

Purpose

Draws a rectangle on the canvas control.

The frame of the rectangle is drawn using the current value of the following attributes.

- ATTR_PEN_COLOR
- ATTR_PEN_MODE
- ATTR_PEN_WIDTH
- ATTR_PEN_STYLE (ignored in Windows when pen width is greater than 1)

The interior of the rectangle is drawn using the current value of the following attributes.

- ATTR_PEN_FILL_COLOR
- ATTR_PEN_MODE
- ATTR_PEN_PATTERN

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A Rect structure specifying the location and size of the rectangle to be drawn.</td>
</tr>
<tr>
<td></td>
<td>drawMode</td>
<td>integer</td>
<td>Specifies whether the rectangle’s frame or interior (or both) are drawn. Valid values: VAL_DRAW_FRAME, VAL_DRAW_INTERIOR, VAL_DRAW_FRAME_AND_INTERIOR</td>
</tr>
</tbody>
</table>
Return Value

| status     | integer | Refer to Appendix A for error codes. |

See also

MakeRect, CanvasDrawRoundedRect

CanvasDrawRoundedRect

```c
int status = CanvasDrawRoundedRect(int panelHandle, int controlID, Rect rect,
                                   int ovalHeight, int ovalWidth,
                                   int drawMode);
```

Purpose

Draws a rounded rectangle on the canvas control. Each corner of the rectangle is drawn as a quadrant of an oval.

The frame of the rectangle is drawn using the current value of the following attributes.

| ATTR_PEN_COLOR
| ATTR_PEN_MODE
| ATTR_PEN_WIDTH
| ATTR_PEN_STYLE  (ignored in Windows when pen width is greater than 1)

The interior of the rectangle is drawn using the current value of the following attributes.

| ATTR_PEN_FILL_COLOR
| ATTR_PEN_MODE
| ATTR_PEN_PATTERN
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A Rect structure specifying the location and size of the rectangle to be drawn.</td>
</tr>
<tr>
<td></td>
<td>ovalHeight</td>
<td>integer</td>
<td>The vertical diameter of the oval whose quadrants are drawn at the corners of the rounded rectangle.</td>
</tr>
<tr>
<td></td>
<td>ovalWidth</td>
<td>integer</td>
<td>The horizontal diameter of the oval whose quadrants are drawn at the corners of the rounded rectangle.</td>
</tr>
<tr>
<td></td>
<td>drawMode</td>
<td>integer</td>
<td>Specifies whether the rectangle’s frame or interior (or both) are drawn. Valid values: VAL_DRAW_FRAME, VAL_DRAW_INTERIOR, VAL_DRAW_FRAME_AND_INTERIOR</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect, CanvasDrawRect
CanvasDrawText

int status = CanvasDrawText(int panelHandle, int controlID, char text[],
char metaFont[], Rect bounds, int alignment);

Purpose

Draws a text string within a specified rectangular area on the canvas control. You can set the
alignment of the string within the rectangle. If the string exceeds the size of the rectangle, it is
clipped.

The text is drawn using the current value of the following attribute.

ATTR_PEN_COLOR

The background rectangle is drawn using the current value of the following attributes:

ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN

If you do not want the background rectangle to be drawn, set the ATTR_PEN_FILL_COLOR
attribute of the canvas control to VAL_TRANSPARENT.

Parameters

| Input  | panelHandle | integer | The specifier for a particular panel that is contained in
memory. This handle will have been returned by the
LoadPanel, NewPanel, or DuplicatePanel
function. |
|--------|-------------|---------|-------------------------------------------------------------|
| controlID | integer | The defined constant (located in the .uir header file)
which was assigned to the control in the User Interface
Editor, or the ID returned by the NewCtrl or
DuplicateCtrl function. |
| text | string | The text string to be drawn within the rectangle. |
| metaFont | string | Specifies the text font. Must be one of the predefined
metafonts (see Table 3-5 in this manual) or a metafont
created by a call to CreateMetaFont. |
| bounds | Rect | A Rect structure specifying location and size of the
background rectangle within which the text is drawn. |
| alignment | integer | Determines the placement of the text string within the
background rectangle. See discussion below. |
Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

Parameter Discussion

The values in the **bounds** parameter are in terms of pixel coordinates, with the origin (0,0) at the upper left corner of the canvas control.

If you want the size of the background rectangle to be adjusted automatically to the display size of the text string, set **height** and **width** in the **bounds** parameter to **VAL_KEEP_SAME_SIZE**.

The valid values for the **alignment** parameters are listed in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_LOWER_LEFT</td>
<td>Draw the string in the lower left corner of the background rectangle.</td>
</tr>
<tr>
<td>VAL_CENTER_LEFT</td>
<td>Start the string from the midpoint of the left edge of the background rectangle.</td>
</tr>
<tr>
<td>VAL_UPPER_LEFT</td>
<td>Draw the string in the upper left corner of the background rectangle.</td>
</tr>
<tr>
<td>VAL_LOWER_CENTER</td>
<td>Center the string just above the bottom edge of the background rectangle.</td>
</tr>
<tr>
<td>VAL_CENTER_CENTER</td>
<td>Center the string in the middle of the background rectangle.</td>
</tr>
<tr>
<td>VAL_UPPER_CENTER</td>
<td>Center the string just below the top edge of the background rectangle.</td>
</tr>
<tr>
<td>VAL_LOWER_RIGHT</td>
<td>Draw the string in the lower right corner of the background rectangle.</td>
</tr>
<tr>
<td>VAL_CENTER_RIGHT</td>
<td>Draw the string so that it ends just at the midpoint of the right edge of the background rectangle.</td>
</tr>
<tr>
<td>VAL_UPPER_RIGHT</td>
<td>Draw the string in the upper right corner of the background rectangle.</td>
</tr>
</tbody>
</table>

If the background rectangle specified by **bounds** is smaller than the text display size, the text is clipped to the rectangle and the specified **alignment** is ignored. If the rectangle width is smaller than the text display width, the text is displayed from the left. If the rectangle height is smaller than the text display height, the text is displayed from the top.

See also

MakeRect, CanvasDrawTextAtPoint
CanvasDrawTextAtPoint

```c
int status = CanvasDrawTextAtPoint(int panelHandle, int controlID, char text[], char metaFont[], Point anchorPoint, int alignment);
```

**Purpose**

Draws a text string at the specified location in the canvas control. The location is in terms of an anchor point and an alignment around the point.

The text is drawn using the current value of the following attribute.

ATTR_PEN_COLOR

The background of the text is drawn using the current value of the following attributes:

ATTR_PEN_FILL_COLOR
ATTR_PEN_MODE
ATTR_PEN_PATTERN

If you do not want the background rectangle to be drawn, set the ATTR_PEN_FILL_COLOR attribute of the canvas control to VAL_TRANSPARENT.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>string</td>
<td>The text string to be drawn at the anchor point.</td>
</tr>
<tr>
<td></td>
<td>metaFont</td>
<td>string</td>
<td>Specifies the text font. Must be one of the predefined metafonts (see Table 3-5 in this manual) or a metafont created by a call to CreateMetaFont.</td>
</tr>
<tr>
<td></td>
<td>anchorPoint</td>
<td>Point</td>
<td>A Point structure specifying location of the point at which the text is drawn.</td>
</tr>
<tr>
<td></td>
<td>alignment</td>
<td>integer</td>
<td>Determines the placement of the text string in relation to the anchor point. See discussion below.</td>
</tr>
</tbody>
</table>
Return Value

| status     | integer | Refer to Appendix A for error codes. |

Parameter Discussion

Each alignment value refers to a point on the rectangle that implicitly encloses the text string. The text string is placed so that the point indicated by the alignment parameter is at the location specified the anchorPoint parameter. The valid values for the alignment parameter are listed in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALLOWER_LEFT</td>
<td>Draw the string so that lower left corner of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALCENTER_LEFT</td>
<td>Draw the string so that midpoint of the left edge of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALUPPER_LEFT</td>
<td>Draw the string so that upper left corner of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALLOWER_CENTER</td>
<td>Draw the string so that midpoint of the bottom edge of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALCENTER_CENTER</td>
<td>Draw the string so that center of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALUPPER_CENTER</td>
<td>Draw the string so that midpoint of the top edge of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALLOWER_RIGHT</td>
<td>Draw the string so that lower right corner of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALCENTER_RIGHT</td>
<td>Draw the string so that midpoint of the right edge of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
<tr>
<td>VALUPPER_RIGHT</td>
<td>Draw the string so that upper right corner of its enclosing rectangle is at the location specified by anchorPoint.</td>
</tr>
</tbody>
</table>

See also

MakeRect, CanvasDrawText
CanvasEndBatchDraw

```c
int nestingDepth = CanvasEndBatchDraw(int panelHandle, int controlID);
```

**Purpose**

Ends the batch drawing started with `CanvasStartBatchDraw`.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>panelHandle</strong></td>
<td>integer</td>
<td>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</td>
</tr>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nestingDepth</td>
<td>integer</td>
<td>The number of calls to <code>CanvasStartBatchDraw</code> that have not been matched by calls to <code>CanvasEndBatchDraw</code> (including this call). A negative value indicates that an error occurred. Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>

**See also**

`CanvasStartBatchDraw`
CanvasGetClipRect

```c
int status = CanvasGetClipRect(int panelHandle, int controlID, Rect *clipRect);
```

**Purpose**

Obtains the current clipping rectangle for the canvas control. All drawing operations are restricted to the area in the clipping rectangle. Any drawing outside the clipping rectangle is not shown. Exception: CanvasClear is not restricted to the clipping rectangle.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>Output</td>
<td>clipRect</td>
<td>Rect</td>
<td>The Rect structure into which the location and size of the clipping rectangle are stored. If clipping is disabled (the default state), the height and width values in the structure are set to zero.</td>
</tr>
</tbody>
</table>

**Return Value**

| status    | integer | Refer to Appendix A for error codes. |

**See Also**

CanvasSetClipRect

---

CanvasGetPenPosition

```c
int status = CanvasGetPenPosition(int panelHandle, int controlID, Point *point);
```

**Purpose**

Obtains the current position of the canvas pen.

**Note:** CanvasDrawLineTo is the only canvas drawing function that uses the pen position.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>Output</td>
<td>point</td>
<td>Point</td>
<td>The Point structure into which the current position of the canvas pen is stored.</td>
</tr>
</tbody>
</table>

Return Value

| status      | integer     | Refer to Appendix A for error codes. |

See Also

CanvasSetPenPosition, CanvasDefaultPen, CanvasDrawLineTo

CanvasGetPixel

```c
int status = CanvasGetPixel(int panelHandle, int controlID, Point pixelPoint, int *pixelColor);
```

Purpose

Obtains the color of a single pixel on a canvas control.

Note: The canvas control maintains an internal bitmap reflecting all of the drawing operations (except for drawing operations made while the ATTR_DRAW_POLICY attribute is set to VAL_DIRECT_TO_SCREEN). There are times during which the internal bitmap contains the result of recent drawing operations that have not yet been reflected on the screen. This function obtains the pixel colors from the internal bitmap, not from the screen.
### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>PanelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>pixelPoint</td>
<td>Point</td>
<td>A Point structure indicating the location of a pixel. The location is in terms of unscaled pixel coordinates. The origin (0,0) is the upper left corner of the canvas control.</td>
</tr>
<tr>
<td>Output</td>
<td>pixelColor</td>
<td>integer</td>
<td>The RGB color value of the pixel at the specified point.</td>
</tr>
</tbody>
</table>

### Return Value

| status     | integer | Refer to Appendix A for error codes. |

### See also

MakePoint, CanvasGetPixels

---

### CanvasGetPixels

```c
int status = CanvasGetPixels(int panelHandle, int controlID, Rect rect, int pixelColors[]);
```

### Purpose

Obtains the colors of the pixels in the specified rectangular area of a canvas control.

### Note:

*The canvas control maintains an internal bitmap reflecting all of the drawing operations (except for drawing operations made while the ATTR_DRAW_POLICY attribute is set to VAL_DIRECT_TO_SCREEN). There are times during which the internal bitmap contains the result of recent drawing operations that have not yet been reflected on the screen. This function obtains the pixel colors from the internal bitmap, not from the screen.*
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>The Rect structure specifying the location and size of the rectangular area from which to obtain the pixel colors. The location and size are expressed in terms of unscaled pixel coordinates. The origin (0,0) is the upper left corner of the canvas control. Use VAL_ENTIRE_OBJECT to specify the entire canvas.</td>
</tr>
</tbody>
</table>

Output | pixelColors | integer array | An array of RGB color values of the pixels in the specified rectangle. See discussion below. |

Parameter Discussion

The total number of elements in the pixelColors array should be equal to rect.height * rect.width. The pixel color values are stored in row-major order. For example, if rect has the following values,

| rect.top  | 50 |
| rect.left | 60 |
| rect.height | 20 |
| rect.width | 15 |

then the color of pixel \(\{x = 65, y = 58\}\) is stored in pixel array at the following index.

\[
(y - \text{rect.top}) \times \text{rect.width} + (x - \text{rect.left})
\]

\[
= (58-50)*15 + (65-60)
\]

\[
= 125
\]

When using a rect.width of VAL_TO_EDGE, substitute the following for rect.width in the above formula.

\[
(\text{total width of canvas}) - \text{rect.left}
\]
Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakePoint, CanvasGetPixel

CanvasInvertRect

int status = CanvasInvertRect (int panelHandle, int controlID, Rect rect);

Purpose

Inverts the colors in the specified rectangular area of a canvas control. The colors that result from the inversion are dependent on the operating system. If you invert the same rectangle twice, you are guaranteed to get the original colors back.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A Rect structure which specifies the location and size of the rectangular area in which to invert the colors. Use VAL_ENTIRE_OBJECT to specify the entire canvas.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect
CanvasScroll

```c
int status = CanvasScroll(int panelHandle, int controlID, Rect scrollRect,
                          int scrollAmountInXDirection, int scrollAmountInYDirection);
```

**Purpose**

Scrolls the contents of the specified rectangular area of a canvas control. The area that is exposed by the scrolling is filled using the current value of the `ATTR_PEN_FILL_COLOR` attribute. The contents of the canvas outside the specified rectangular area is not affected by the scrolling.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td>scrollRect</td>
<td>Rect</td>
<td>A <code>Rect</code> structure which specifies the location and size of the rectangular area to scroll. Use <code>VAL_ENTIRE_OBJECT</code> to specify the entire canvas.</td>
</tr>
<tr>
<td>scrollAmountInXDirection</td>
<td>integer</td>
<td>The amount to scroll horizontally.</td>
</tr>
<tr>
<td>scrollAmountInYDirection</td>
<td>integer</td>
<td>The amount to scroll vertically.</td>
</tr>
</tbody>
</table>

**Parameter Discussion**

A positive value for `scrollAmountInXDirection` moves the contents of the rectangle to the right. An area on the left side of the rectangle is thereby exposed. It is filled with the current value of the `ATTR_PEN_FILL_COLOR` attribute.

A negative value for `scrollAmountInXDirection` moves the contents of the rectangle to the left. An area on the right side of the rectangle is thereby exposed. It is filled with the current value of the `ATTR_PEN_FILL_COLOR` attribute.

A positive value for `scrollAmountInYDirection` moves the contents of the rectangle down. An area at the top of the rectangle is thereby exposed. It is filled with the current value of the `ATTR_PEN_FILL_COLOR` attribute.
A negative value for `scrollAmountInYDirection` moves the contents of the rectangle the up. An area at the bottom of the rectangle is thereby exposed. It is filled with the current value of the `ATTR_PEN_FILL_COLOR` attribute.

**See Also**

`MakeRect`

---

**CanvasSetClipRect**

```c
int status = CanvasSetClipRect (int panelHandle, int controlID, Rect clipRect);
```

**Purpose**

Sets the clipping rectangle for the canvas control. All drawing operations are restricted to the area in the clipping rectangle. Any drawing outside the clipping rectangle is not shown. Exception: `CanvasClear` is not restricted to the clipping rectangle.

Changing the clipping rectangle does not affect current contents of the canvas.

In the initial state for a canvas control, clipping is disabled.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td>clipRect</td>
<td>Rect</td>
<td>A <code>Rect</code> structure specifying into the location and size of the clipping rectangle. To disable clipping, set the height and width of <code>clipRect</code> to zero, or use <code>VAL_EMPTY_RECT</code>.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
</tr>
</tbody>
</table>
See Also

CanvasGetClipRect

CanvasSetPenPosition

int status = CanvasSetPenPosition (int panelHandle, int controlID, Point point);

Purpose

Sets the position of the canvas pen.

Note: CanvasDrawLineTo is the only canvas drawing function that uses the pen position.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>point</td>
<td>Point</td>
<td>A Point structure specifying the new position of the canvas pen.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

CanvasGetPenPosition, CanvasDefaultPen, CanvasDrawLineTo
CanvasStartBatchDraw

```c
int nestingDepth = CanvasStartBatchDraw(int panelHandle, int controlID);
```

**Purpose**

This function can be used to increase the drawing performance of the canvas control. In general, it should be used whenever you want to make two or more consecutive calls to canvas drawing functions. Each call to CanvasStartBatchDraw should be matched with a call to CanvasEndBatchDraw.

Before drawing operations can be performed, certain operating system functions must be invoked to prepare for drawing on the particular canvas. Without batch drawing, these system functions must be called for each canvas drawing operation. With batch drawing, the system functions are called only once for all of the drawing operations between CanvasStartBatchDraw and the matching CanvasEndBatchDraw.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

**Return Value**

| nestingDepth | integer | The number of calls to CanvasStartBatchDraw (including this call) that have not been matched by calls to CanvasEndBatchDraw. A negative value indicates that an error occurred. Refer to Appendix A for error codes. |

**Using This Function**

The following code shows an example of how you would incorporate a sequence of drawing operations on the same canvas control into one batch.

```c
CanvasStartBatchDraw (panelHandle, controlID);
CanvasDrawLine (panelHandle, controlID, point1, point2);
CanvasDrawLine (panelHandle, controlID, point3, point4);
CanvasDrawRect (panelHandle, controlID, rect5);
CanvasEndBatchDraw (panelHandle, controlID);
```
During a batch draw, you may call drawing operations on other canvas controls or call other User Interface Library functions that perform drawing operations or process events. This has the effect of implicitly ending the batch. The next time you call a drawing function on the same canvas, the batch is implicitly restarted.

You may nest calls to `CanvasStartBatchDraw`.

Failure to properly match `CanvasStartBatchDraw` and `CanvasEndBatchDraw` calls can negate the potential performance improvements but does not cause any other ill effects.

**Note:** *If theATTR_DRAW_POLICY attribute for the canvas control is set to VAL_UPDATE_IMMEDIATELY, no update to the screen occurs until the batch is ended. Also, changing values of theATTR_DRAW_POLICY and ATTR_OVERLAP_POLICY attributes during a batch draw has no effect until after the batch is ended.*

**See also**

`CanvasEndBatchDraw`

---

**CanvasUpdate**

```c
int status = CanvasUpdate (int panelHandle, int controlID, Rect updateArea);
```

**Purpose**

Immediately updates on the screen the contents of the canvas control within the specified rectangular area.

The canvas control maintains an internal bitmap reflecting all of the drawing operations (except for drawing operations made while the ATTR_DRAW_POLICY attribute is set to VAL_DIRECT_TO_SCREEN). Maintaining the internal bitmap ensures that the canvas is redrawn correctly when it is exposed.

This function copies the content of the specified area in the internal bitmap to the canvas control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>updateArea</td>
<td>Rect</td>
<td>A Rect structure specifying the location and size of the rectangular to be updated from the internal bitmap. Use VAL_ENTIRE_OBJECT to specify the entire canvas control.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See also

MakeRect

---

CheckListItem

```c
int status = CheckListItem(int panelHandle, int controlID, int itemIndex,
                          int checked);
```

Purpose

This function places a check by, or removes a check from, the specified list item. It applies only to list boxes with the check mode attribute set.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list where the item will be placed.</td>
</tr>
<tr>
<td></td>
<td>checked</td>
<td>integer</td>
<td>Specifies whether or not the specified list item is checked.</td>
</tr>
</tbody>
</table>
Return Value

| status | integer | Refer to Appendix A for error codes. |

---

**ClearAxisItems**

```c
int status = ClearAxisItems(int panelHandle, int controlID, int axis);
```

**Purpose**

This function deletes all string/value pairs from the list of label strings for a graph or strip chart axis.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td><strong>axis</strong></td>
<td>integer</td>
<td>Specifies the axis from which to delete all of the string/value pair(s). Valid values: VAL_XAXIS, VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS (graphs only)</td>
</tr>
</tbody>
</table>
Return Value

| status        | integer | Refer to Appendix A for error codes. |

See Also

InsertAxisItem, ReplaceAxisItem, DeleteAxisItem, GetNumAxisItems

---

ClearListCtrl

```c
int status = ClearListCtrl(int panelHandle, int controlID);
```

**Purpose**

This function clears all list items from the specified list control.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

---

ClearStripChart

```c
int status = ClearStripChart(int panelHandle, int controlID);
```

**Purpose**

Clears all traces from a strip chart control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

ClipboardGetBitmap

```c
int status = ClipboardGetBitmap(int *bitmapID, int *available);
```

Purpose

Determines whether or not a bitmap image is available on the system clipboard and optionally retrieves a copy of the bitmap. The bitmap ID can then be passed to any function that accepts a bitmap, such as CanvasDrawBitmap.

You can discard the bitmap by passing its ID to DiscardBitmap.

Parameters

<table>
<thead>
<tr>
<th>Output</th>
<th>bitmapID</th>
<th>integer</th>
<th>An ID that serves as a handle to the bitmap copied from the clipboard. It is set to NULL if there is no bitmap on the clipboard. If you not want a copy of the bitmap, pass NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>available</td>
<td>integer</td>
<td>Is set to 1 if a bitmap is available on the system clipboard, 0 otherwise. You may pass NULL for this parameter.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
See Also

ClipboardPutBitmap, ClipboardGetText, GetBitmapData, SetCtrlBitmap, PlotBitmap, CanvasDrawBitmap, DiscardBitmap.

---

**ClipboardGetText**

```c
int status = ClipboardGetText(char **text, int *available);
```

**Purpose**

Determines whether or not a text string is available on the system clipboard and optionally retrieves a copy of the text.

When the copy of the text is no longer needed, you should free it by calling the `free` function.

**Parameters**

| Output | text          | string | A pointer to the nul-terminated string copied from the clipboard. It is set to NULL if there is no text on the clipboard. If you not want a copy of the text, pass NULL.
|        | available     | integer| Is set to 1 if a text string is available on the system clipboard, 0 otherwise. You may pass NULL for this parameter.

**Return Value**

| status | integer | Refer to Appendix A for error codes.

**See Also**

ClipboardPutText, ClipboardGetBitmap.

---

**ClipboardPutBitmap**

```c
int status = ClipboardPutBitmap(int bitmapID);
```

**Purpose**

Copies a bitmap image onto the system clipboard.
Parameter

| Input | bitmapID | integer | The ID of the bitmap object containing the image. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, or GetPanelDisplayBitmap. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also


---

ClipboardPutText

```c
int status = ClipboardPutText (char text[]);
```

Purpose

Copies a text string onto the system clipboard.

Parameter

| Input | text | string | A nul-terminated string. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

ClipboardGetText, ClipboardPutBitmap
ConfigurePrinter

```c
int status = ConfigurePrinter(char printFile[], int orientation, int imageWidth,
                               int imageHeight, int ejectAfter);
```

**Purpose**

Sets the orientation, height, width, and eject after print attributes. See Table 3-27, Hard Copy Attributes, in Chapter 3, Programming with the User Interface Library, for more information on printing attributes.

This function is retained for compatibility with LabWindows for DOS. It is superseded by SetPrintAttribute.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>printFile</th>
<th>string</th>
<th>Is ignored in LabWindows/CVI because the print file is specified at the system level.</th>
</tr>
</thead>
<tbody>
<tr>
<td>orientation</td>
<td>integer</td>
<td></td>
<td>The orientation of the image on the page. Valid Value: 1 = VAL_PORTRAIT, 2 = VAL_LANDSCAPE</td>
</tr>
<tr>
<td>imageWidth</td>
<td>integer</td>
<td></td>
<td>The width of the image in millimeter/10 or VAL_USE_PRINTER_DEFAULT or VAL_INTEGRAL_SCALE</td>
</tr>
<tr>
<td>imageHeight</td>
<td>integer</td>
<td></td>
<td>The height of the image in millimeter/10 or VAL_USE_PRINTER_DEFAULT or VAL_INTEGRAL_SCALE</td>
</tr>
<tr>
<td>ejectAfter</td>
<td>integer</td>
<td></td>
<td>Determines if the next output will be ejected from the printer. While ejectAfter is set to zero, outputs will print on the same page until ejectAfter is set to one.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

ConfirmPopup

```c
int status = ConfirmPopup(char title[], char message[]);
```

**Purpose**

Displays a prompt message in a dialog box and waits for the user to select the Yes or No button.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>title</th>
<th>string</th>
<th>The title of the dialog box.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>message</td>
<td>string</td>
<td>The message displayed on the dialog box.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

Return Codes

| 1     | user selected Yes. |
| 0     | user selected No.  |

CreateMetaFont

```c
int status = CreateMetaFont (char newMetaFontName[], char existingFontName[], int pointSize, int bold, int italics, int underlined, int strikeout);
```

Purpose

Creates a new meta font based on a pre-defined National Instruments font, an existing metafont, or a font supplied by the operating system. Metafonts contain typeface information, point size, and text style.
## Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>newMetaFontName</th>
<th>string</th>
<th>The name to be associated with the new meta font.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>existingFontName</td>
<td>string</td>
<td>The name of the existing font on which the new meta font is based. The font may be one of the National Instrument fonts, a user-defined font saved by a previous <code>CreateMetaFont</code> function call, or a font supplied by the operating system.</td>
</tr>
<tr>
<td></td>
<td>pointSize</td>
<td>integer</td>
<td>The point size of the new meta font. Any positive integer value is valid.</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>integer</td>
<td>Indicates whether or not the newly created meta font will have bold text. 0 = not bold 1 = bold</td>
</tr>
<tr>
<td></td>
<td>italics</td>
<td>integer</td>
<td>Indicates whether or not the newly created meta font will have italicized text. 0 = not italics 1 = italics</td>
</tr>
<tr>
<td></td>
<td>underlined</td>
<td>integer</td>
<td>Indicates whether or not the newly created meta font will have underlined text. 0 = not underlined 1 = underlined</td>
</tr>
<tr>
<td></td>
<td>strikeout</td>
<td>integer</td>
<td>Indicates whether or not the newly created meta font will have strikeout text. 0 = not strikeout 1 = strikeout</td>
</tr>
</tbody>
</table>

## Return Value

| status | integer | Refer to Appendix A for error codes. |

## Parameter Discussion

The following are examples of values that can be passed to `existingFontName`.

- `VAL_7SEG_META_FONT`
- `VAL_DIALOG_FONT`
- "Courier"
DefaultCtrl

```c
int status = DefaultCtrl (int panelHandle, int controlID);
```

**Purpose**

This function restores a control to its default value.

If the control is visible, it is updated to reflect its default value. You assign default values to controls in the User Interface Editor or through the `SetCtrlAttribute` function.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>panelHandle</code></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>controlID</code></td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

DefaultPanel

```c
int status = DefaultPanel (int panelHandle);
```

**Purpose**

Restores all panel controls to their default values.

If the panel is visible, it is updated to reflect the new control values. You assign default values to controls in the User Interface Editor or through the `SetCtrlAttribute` function using `ATTR_DFLT_VALUE`. 
Parameters

| Input | panelHandle | integer | The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

DeleteAxisItem

```c
int status = DeleteAxisItem(int panelHandle, int controlID, int axis, int itemIndex, int numberOfItems);
```

Purpose

Deletes one or more string/value pairs from the list of label strings for a graph or strip chart axis. These strings appear in place of the numerical labels. They appear at the location of their associated values on the graph or strip chart.

To see string labels on an X axis, you must set the ATTR_XUSE_LABEL_STRINGS attribute to TRUE. To see string labels on a Y axis, you must set the ATTR_YUSE_LABEL_STRINGS attribute to TRUE.

Parameters

| Input | panelHandle | integer | The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. |
| controlID | integer | The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function. |
| axis | integer | Specifies the axis from which to delete the selected string/value pair(s). Valid values: VAL_XAXIS VAL_LEFT_YAXIS VAL_RIGHT_YAXIS (graphs only) |
| itemIndex | integer | The zero-based index of the first item to be deleted. |
| numberOfItems | string | The number of items to delete. |
DeleteGraphPlot

\[
\text{int } \text{status} = \text{DeleteGraphPlot}(\text{int } \text{panelHandle}, \text{int } \text{controlID}, \text{int } \text{plotHandle}, \\
\hspace{1em} \text{int } \text{refresh});
\]

**Purpose**

Deletes one or all plots from a graph control.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for the particular plot to delete or -1 to delete all plots in the graph control.</td>
</tr>
<tr>
<td></td>
<td>refresh</td>
<td>integer</td>
<td>Selects when to refresh the graph control. The following choices are valid: VAL_DELAYED_DRAW Delayed Draw. VAL_IMMEDIATE_DRAW Immediate Draw. VAL_NO_DRAW No Draw.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
<td>Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>
Parameter Discussion

**plotHandle** will have been returned by one of the following functions.

PlotX
PlotY
PlotXY
PlotWaveform
PlotPoint
PlotText
PlotLine
PlotRectangle
PlotPolygon
PlotOval
PlotArc
PlotIntensity
PlotBitmap

If **refresh** is set to Delayed Draw, the deleted plot remains on the graph until one of the following actions takes place.

- Graph is rescaled.
- Size of the plot area is changed.
- Graph is overlapped.
- Graph is made visible after it was hidden.
- **ATTR_REFRESH_GRAPH** is set to 1 when it was previously 0.
- **RefreshGraph** function is called.
- Another plot is directed to the graph while **ATTR_REFRESH_GRAPH** is 1.

When one of the above events occurs, the entire plot area is redrawn.

If **refresh** is set to **VAL_IMMEDIATE_DRAW**, the plot area is redrawn immediately and the deleted plot is removed from the graph.

If **refresh** is set to **VAL_NO_DRAW**, the deleted plot remains on the graph until one of the following actions takes place.

- Graph is rescaled.
- Size of the plot area is changed.
- Graph is overlapped when **ATTR_SMOOTH_UPDATE** is 0.
- Graph is made visible when **ATTR_SMOOTH_UPDATE** is 0 after it was hidden.
DeleteImage

```c
int status = DeleteImage (int panelHandle, int controlID);
```

**Purpose**

Removes an image from the specified picture control and from memory.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

DeleteListItem

```c
int status = DeleteListItem (int panelHandle, int controlID, int itemIndex, int numberOfItems);
```

**Purpose**

Deletes one or more items from a list control starting at Item Index.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index of the first item to be deleted.</td>
</tr>
<tr>
<td></td>
<td>numberOfItems</td>
<td>integer</td>
<td>The number of items to delete. To delete all items from itemIndex to the end, enter -1 for numberOfItems.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

DeleteTextBoxLine

```
int status = DeleteTextBoxLine(panelHandle, controlID, lineIndex);
```

Purpose

This function removes the line of text at the specified index.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>lineIndex</td>
<td>integer</td>
<td>The zero-based index of the text box line to be deleted.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
DirSelectPopup

```c
int status = DirSelectPopup(char defaultDirectory[], char title[], int allowCancel,
                            int allowMakeDirectory, char pathName[]);
```

**Purpose**

Displays a file-selection dialog box and waits for the user to select a directory or cancel.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>defaultDirectory</th>
<th>string</th>
<th>The initial directory. If &quot;&quot; is entered, then the current working directory is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>title</td>
<td>string</td>
<td>The title of the dialog box.</td>
</tr>
<tr>
<td>allowCancel</td>
<td>allowCancel</td>
<td>integer</td>
<td>If non-zero, the user can cancel out of the dialog box. If zero, the user must make a selection.</td>
</tr>
<tr>
<td>allowMakeDirectory</td>
<td>allowMakeDirectory</td>
<td>integer</td>
<td>If non-zero, the user is allowed to create a new directory.</td>
</tr>
</tbody>
</table>

| Output     | pathName         | string | The buffer in which the user's selection is returned. The buffer must be at least MAX_PATHNAME_LEN bytes long. |

**Return Value**

| status     | integer | Refer to Appendix A for error codes. |

**Return Codes**

<table>
<thead>
<tr>
<th>0</th>
<th>VAL_NO_DIRECTORY_SELECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VAL_DIRECTORY_SELECTED</td>
</tr>
</tbody>
</table>

**Parameter Discussion**

The maximum length of `defaultDirectory` is MAX_PATHNAME_LEN characters, including the ASCII NUL byte.

The maximum length of `title` is 255 excluding the ASCII NUL byte.
**DiscardBitmap**

```c
int status = DiscardBitmap (int bitmapID);
```

**Purpose**

Discards a bitmap object.

**Parameter**

<table>
<thead>
<tr>
<th>Input</th>
<th>bitmapID</th>
<th>integer</th>
<th>The ID of the bitmap object. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, or GetPanelDisplayBitmap.</th>
</tr>
</thead>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

**See Also**


---

**DiscardCtrl**

```c
int status = DiscardCtrl (int panelHandle, int controlID);
```

**Purpose**

This function removes a control from the specified panel and from memory.

When discarding a control from its own callback function, you can call DiscardCtrl only when responding to the EVENT_COMMIT event. Discarding the control from its own callback function in response to other events may cause unpredictable behavior.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

DiscardMenu

```c
int status = DiscardMenu(int menuBarHandle, int menuID);
```

Purpose

This function removes the specified menu and its sub-menus and items from the menu bar.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar. The ID for a particular menu within a menu bar. The Menu ID should be a constant name (located in the .uir header file) generated by the User Interface Editor or a value returned by the NewMenu function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>menuID</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |


DiscardMenuBar

```c
int status = DiscardMenuBar(int menuBarHandle);
```

**Purpose**

Removes the specified menu bar from every panel on which it resides and from memory.

**Parameters**

| Input   | menuBarHandle | integer | The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar. |

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

DiscardMenuItem

```c
int status = DiscardMenuItem(int menuBarHandle, int menuItemID);
```

**Purpose**

This function removes a menu item from the specified menu and from memory.

**Parameters**

| Input   | menuBarHandle | integer | The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar. |
|         | menuItemID    | integer | The ID used to reference this menu item. The Menu Item ID will have been generated by the User Interface Editor or returned by the NewMenuItem function. |

**Return Value**

| status | integer | Refer to Appendix A for error codes. |
DiscardPanel

```c
int status = DiscardPanel (int panelHandle);
```

**Purpose**

Removes a panel from memory and clears it from the screen if visible.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

DiscardSubMenu

```c
int status = DiscardSubMenu (int menuBarHandle, int subMenuID);
```

**Purpose**

This function removes a submenu from the specified menu bar and from memory.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>subMenuID</td>
<td>integer</td>
<td>The ID for a particular sub-menu within a menu bar. The <code>subMenuID</code> will have been a constant name generated by the User Interface Editor or a value returned by the NewSubMenuItem function.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>
DisplayImageFile

int status = DisplayImageFile (int panelHandle, int controlID, char filename[]);

Purpose

Displays the contents of an image file on a specified picture control.

Supported image types:

PCX: Windows and UNIX
BMP, DIB, RLE, ICO: Windows only
WMF: Windows 95 and NT only
XWD: UNIX only

To display an image, first create a picture control in the User Interface Editor or with the NewCtrl function. Then call the DisplayImageFile function using the control ID.

To delete the image, call the DeleteImage function (which also removes it from memory.)

To replace the current image with another, simply call DisplayImageFile for the same picture control with a different image file.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>filename</td>
<td>string</td>
<td>The name of the image file which contains the image. If the name is a simple filename (in other words, contains no directory path), then the file is loaded from the project. If it is not found in the project, the file is loaded from the directory containing the project.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
**DisplayPCXFile**

```c
int controlID = DisplayPCXFile(char filename[], int controlTop, int controlLeft);
```

**Purpose**

Displays the contents of an image file in a new picture control on the DOS Compatibility window at the given x and y coordinates. Window coordinates may be any value between 0 and 32767, with the origin in the upper left corner directly beneath the title bar before the panel is scrolled. These picture controls will be overlapped by child panels of this window. If the DOS Compatibility window has not yet been created, `DisplayPCXFile` will create it.

**Supported image types:**

- **PCX:** Windows and UNIX
- **BMP, DIB, RLE, ICO:** Windows only
- **WMF:** Windows 95 and NT only
- **XWD:** UNIX only

**Note:** *This function has been replaced by DisplayImageFile, and should be translated by the user. Repeated use of this function will clutter the panel with redundant controls.*

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>filename</th>
<th>string</th>
<th>The name of the image file which contains the image. If the name is a simple filename (in other words, contains no directory path), then the file is loaded from the project. If it is not found in the project, the file is loaded from the directory containing the project.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlTop</td>
<td>integer</td>
<td>The coordinate of the top edge of the image.</td>
</tr>
<tr>
<td></td>
<td>controlLeft</td>
<td>integer</td>
<td>The coordinate of the left edge of the image.</td>
</tr>
</tbody>
</table>

**Return Value**

| controlID  | integer | Returns the ID used to specify this control in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes. |

---

**DisplayPanel**

```c
int status = DisplayPanel(int panelHandle);
```

**Purpose**

Displays a panel on the screen.
When a panel is visible and not dimmed, the user can operate it and events can be generated from it. Calling `DisplayPanel` when a panel is already displayed will cause the panel to be completely redrawn.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>panelHandle</code></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
</table>

**Return Value**

<table>
<thead>
<tr>
<th><code>status</code></th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

**DOSColorToRGB**

```c
int status = DOSColorToRGB(int lwDOSColor);
```

**Purpose**

Translates the 16 standard color values from LabWindows/DOS to RGB values as shown in the following table.

<table>
<thead>
<tr>
<th>LW/DOS</th>
<th>LW/CVI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (black)</td>
<td>VAL_BLACK = 0x00000000L</td>
</tr>
<tr>
<td>1 (dark blue)</td>
<td>VAL_DK_BLUE = 0x00008000L</td>
</tr>
<tr>
<td>2 (dark green)</td>
<td>VAL_DK_GREEN = 0x00800000L</td>
</tr>
<tr>
<td>3 (dark cyan)</td>
<td>VAL_DK_CYAN = 0x00808000L</td>
</tr>
<tr>
<td>4 (dark red)</td>
<td>VAL_DK_RED = 0x80000000L</td>
</tr>
<tr>
<td>5 (dark magenta)</td>
<td>VAL_DK_MAGENTA = 0x80008000L</td>
</tr>
<tr>
<td>6 (brown)</td>
<td>VAL_DK_YELLOW = 0x80800000L</td>
</tr>
<tr>
<td>7 (gray)</td>
<td>VAL_LT_GRAY = 0xC0C0C0C0L</td>
</tr>
<tr>
<td>8 (dark gray)</td>
<td>VAL_DK_GRAY = 0x80808080L</td>
</tr>
<tr>
<td>9 (blue)</td>
<td>VAL_BLUE = 0x000000FFL</td>
</tr>
</tbody>
</table>

(continues)
(Continued)

<table>
<thead>
<tr>
<th>LW/DOS</th>
<th>LW/CVI</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (green)</td>
<td>VAL_GREEN = 0x00FF00L</td>
</tr>
<tr>
<td>11 (cyan)</td>
<td>VAL_CYAN = 0x00FFFFL</td>
</tr>
<tr>
<td>12 (red)</td>
<td>VAL_RED = 0xFF0000L</td>
</tr>
<tr>
<td>13 (magenta)</td>
<td>VAL_MAGENTA = 0xFF00FFL</td>
</tr>
<tr>
<td>14 (yellow)</td>
<td>VAL_YELLOW = 0xFFFF00L</td>
</tr>
<tr>
<td>15 (white)</td>
<td>VAL_WHITE = 0xFFFFFFFF</td>
</tr>
</tbody>
</table>

Parameters

| Input  | lwDOSColor | integer | One of the 16 standard colors from LabWindows/DOS. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

DOSCompatWindow

```c
int panelHandle = DOSCompatWindow(void);
```

Purpose

Displays a window which serves as a parent panel for DOS LabWindows panels.

A panel handle is returned that is used by the parentPanelHandle parameter in the Load Panel function. This function can be used with DOS LabWindows .uir files to retain the appearance of the DOS User Interface screen.

This function exists as an aid for converting LabWindows for DOS applications. It should not be used when developing new applications in LabWindows/CVI.

Return Value

| panelHandle | integer | Returns an integer specifier (panel handle) for the DOS Compatibility Window. A negative number indicates that an error occurred. Refer to Appendix A for error codes. |

LabWindows/CVI User Interface Reference 4-68 © National Instruments Corporation
DuplicateCtrl

```c
int newID = DuplicateCtrl(int sourcePanelHandle, int controlID,
                          int destinationPanelHandle, char duplicateLabel[],
                          int controlTop, int controlLeft);
```

**Purpose**

This function copies an existing control from the source panel to the destination panel and returns a control ID which is used to reference the control in subsequent function calls.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcePanelHandle</td>
<td>integer The handle of the source panel containing the control to be duplicated. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>destinationPanelHandle</td>
<td>integer Specifies the parent panel handle into which the duplicate control is copied. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>duplicateLabel</td>
<td>string The label for the duplicate control. Pass &quot;&quot; for no label. Pass 0 to use the label of the source control.</td>
</tr>
<tr>
<td>controlTop</td>
<td>integer The vertical coordinate at which the upper left corner of the control (not including labels) is placed.</td>
</tr>
<tr>
<td>controlLeft</td>
<td>integer The horizontal coordinate at which the upper left corner of the control (not including labels) is placed.</td>
</tr>
</tbody>
</table>

**Return Value**

| newID | integer | Returns the ID used to specify the new (duplicate) control in subsequent function calls. Refer to Appendix A for error codes. |
Parameter Discussion

The valid range for `controlTop` and `controlLeft` is -32,768 to 32,767 or `VAL_KEEP_SAME_POSITION`. The origin (0,0) is at the upper-left corner of the panel (directly below the title bar) before the panel is scrolled.

DuplicatePanel

```
int status = DuplicatePanel (int destParentPanelHandle, int originalPanelHandle,
                          char duplicatePanelTitle[], int panelTop, int panelLeft);
```

Purpose

This function duplicates a panel into the specified destination parent panel and returns the duplicate (new) panel handle.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destParentPanelHandle</td>
<td>integer The handle for the parent panel into which the duplicate panel is copied. To make the panel a top-level panel, enter 0.</td>
</tr>
<tr>
<td>originalPanelHandle</td>
<td>integer The handle of the original panel to be duplicated. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>duplicatePanelTitle</td>
<td>string The title of the duplicate (new) panel. Pass &quot;&quot; for no title. Pass 0 to use the same title as the original panel.</td>
</tr>
<tr>
<td>panelTop</td>
<td>integer The vertical coordinate at which the upper left corner of the panel (directly below the title bar) is placed. Pass VAL_KEEP_SAME_POSITION to keep the same top coordinate as the original panel.</td>
</tr>
<tr>
<td>panelLeft</td>
<td>integer The horizontal coordinate at which the upper left corner of the panel (directly below the title bar) is placed. Pass VAL_KEEP_SAME_POSITION to keep the same left coordinate as the original panel.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
Parameter Discussion

For a top-level panel, (0,0) is the upper-left corner of the screen. For a child panel, (0,0) is the upper-left corner of the parent panel (directly below the title bar) before the parent panel is scrolled. The panelTop and panelLeft coordinates must be integer values from -32768 to 32767, or VAL_AUTO_CENTER to center the panel.

EmptyMenu

```c
int status = EmptyMenu(int menuBarHandle, int menuID);
```

Purpose

This function removes all sub-menus and menu items from the specified menu.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.

The ID for a particular menu within a menu bar. The Menu ID should be a constant name (located in the .uir header file) generated by the User Interface Editor or a value returned by the NewMenu function.

Return Value

| status | integer | Refer to Appendix A for error codes.

EmptyMenuBar

```c
int status = EmptyMenuBar(int menuBarHandle);
```

Purpose

This function removes all menus and menu items from the specified menu bar but retains the menu bar handle in memory.
Parameters

| Input  | menuBarHandle | integer | The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

FakeKeystroke

```c
int status = FakeKeystroke(int keyCode);
```

Purpose

Simulates a keystroke.

Note: The order in which the keystroke is received in relation to other events is not defined, so you must use this function with care.

Parameters

| Input  | keyCode | integer | Key codes are formed by a logical OR operation on values representing a modifier key and either an ASCII character or a virtual key. See Table 3-7 in Chapter 3 for modifier and virtual keys. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

FileSelectPopup

```c
int status = FileSelectPopup(char defaultDirectory[], char defaultFileSpec[], char fileTypeList[], char title[], int buttonLabel, int restrictDirectory, int restrictExtension, int allowCancel, int allowMakeDirectory, char pathName[]);
```

Purpose

Displays a file-selection dialog box and waits for the user to select a file or cancel.
## Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>defaultDirectory</strong></td>
<td>string  The initial directory. If &quot;&quot; is entered, then the current working directory will be used.</td>
</tr>
<tr>
<td><strong>defaultFileSpec</strong></td>
<td>string  A string that specifies which files to display. For example, &quot;*.c&quot; would cause all files with the extension .c to be displayed.</td>
</tr>
<tr>
<td><strong>fileTypeList</strong></td>
<td>string  A list of file types, separated by semicolons, to be contained in the File Type List of the File Select Pop-up when <strong>restrictExtension</strong> is FALSE. For example, &quot;<em>.c;</em>.h&quot; allows the user to select &quot;<em>.c&quot; or &quot;</em>.h&quot; from the File Type List. (&quot;<em>.</em>&quot; is always available).</td>
</tr>
<tr>
<td><strong>title</strong></td>
<td>string  The title of the dialog box.</td>
</tr>
</tbody>
</table>
| **buttonLabel** | integer  Selects the label for the file select button. The choices are:  
"OK" = VAL_OK_BUTTON  
"Save" = VAL_SAVE_BUTTON  
"Select" = VAL_SELECT_BUTTON  
(affects existing files only)  
"Load" = VAL_LOAD_BUTTON (affects existing files only) . |
| **restrictDirectory** | integer  If non-zero, the user cannot change directories or drives. If zero, the user can change directories or drives. |
| **restrictExtension** | integer  If non-zero, the user is limited to files with the default extension. If zero, the user can select files with any extension. |
| **allowCancel** | integer  If non-zero, the user can cancel out of the File Select Popup. If zero, the user can leave the pop-up only by making a selection. |
| **allowMakeDirectory** | integer  If non-zero, allows the user to make a new directory from the File Select Popup. This is useful when a user wants to save a file into a new directory. |

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pathName</strong></td>
<td>string  The buffer in which the user's selection is returned. The buffer must be at least MAX_PATHNAME_LEN bytes long.</td>
</tr>
</tbody>
</table>

## Return Value

| status | integer  Refer to Appendix A for error codes. |
Return Codes

<table>
<thead>
<tr>
<th></th>
<th>Return Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VAL_NO_FILE_SELECTED</td>
</tr>
<tr>
<td>1</td>
<td>VAL_EXISTING_FILE_SELECTED</td>
</tr>
<tr>
<td>2</td>
<td>VAL_NEW_FILE_SELECTED</td>
</tr>
</tbody>
</table>

Parameter Discussion

The `defaultfilespec` is shown in the file name control when the pop-up is first displayed. If you specify an actual file name, such as `test.c`, that name appears in the file name box and also in the file list box. The default file specification (spec) cannot contain a directory path.

You need to declare the `pathName` string to be at least `MAX_PATHNAME_LEN` bytes long.

FontSelectPopup

```c
int status = FontSelectPopup (char title[], char sampleText[],
    int monospacedFontsOnly, char typeName[],
    int *bold, int *underline, int *strikeOut, int *italic,
    int *justification, int *textColor, int *fontSize,
    int minimumFontSize, int maximumFontSize,
    int showDefaultButton, int allowMetaFonts);
```

Brings up a dialog which allows the user to specify font settings.
## Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>title</td>
<td>string</td>
<td>The title to be displayed on the dialog box. The maximum length is 255 characters.</td>
</tr>
<tr>
<td>Input</td>
<td>sampleText</td>
<td>string</td>
<td>The sample text to be displayed in the font select pop-up as demonstration of how the settings will affect the appearance of text.</td>
</tr>
<tr>
<td>Input</td>
<td>monospacedFontsOnly</td>
<td>integer</td>
<td>If nonzero, the user can select only monospaced (fixed width) fonts. If zero, the user may select any font.</td>
</tr>
<tr>
<td>Input</td>
<td>minimumFontSize</td>
<td>integer</td>
<td>The minimum value allowed in the 'Font Size' control.</td>
</tr>
<tr>
<td>Input</td>
<td>maximumFontSize</td>
<td>integer</td>
<td>The maximum value allowed in the 'Font Size' control.</td>
</tr>
<tr>
<td>Input</td>
<td>showDefaultButton</td>
<td>integer</td>
<td>If zero, the default button is hidden. If non-zero, the default button is shown. When the default button is pushed, the value of every control on the pop-up is set to the values specified in the last call to SetFontPopupDefaults.</td>
</tr>
<tr>
<td>Input</td>
<td>allowMetaFonts</td>
<td>integer</td>
<td>If zero, the National Instruments supplied metafonts are not listed in the typeface selection ring. If nonzero, the metafonts are listed.</td>
</tr>
<tr>
<td>Input/Output</td>
<td>typefaceName</td>
<td>string</td>
<td>On input, this buffer contains the typeface name (for example, &quot;Courier&quot;) to be initially displayed in the selection ring. On output, this buffer contains the typeface name the user has selected. The buffer must be at least 256 bytes long. Pass 0 to hide the typeface selection ring and prevent the user from changing the typeface.</td>
</tr>
<tr>
<td>Input/Output</td>
<td>bold</td>
<td>integer</td>
<td>On input, the value to initially set in the 'Bold' checkbox. On output, the final value in the 'Bold' checkbox. Pass 0 to hide the 'Bold' checkbox.</td>
</tr>
</tbody>
</table>

(continues)
Parameters (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>underline</td>
<td>integer</td>
<td>On input, the value to initially set in the 'Underline' checkbox. On output, the final value in the 'Underline' checkbox. Pass 0 to hide the 'Underline' checkbox.</td>
</tr>
<tr>
<td>strikeOut</td>
<td>integer</td>
<td>On input, the value to initially set in the 'StrikeOut' checkbox. On output, the final value in the 'StrikeOut' checkbox. Pass 0 to hide the 'StrikeOut' checkbox.</td>
</tr>
<tr>
<td>italic</td>
<td>integer</td>
<td>On input, the value to initially set in the 'Italic' checkbox. On output, the final value in the 'Italic' checkbox. Pass 0 to hide the 'Italic' checkbox.</td>
</tr>
<tr>
<td>justification</td>
<td>integer</td>
<td>On input, the value to initially set in the 'Justification' ring control. On output, the final value in the 'Justification' ring control. Pass 0 to hide the 'Justification' ring control.</td>
</tr>
<tr>
<td>textColor</td>
<td>integer</td>
<td>On input, the value to initially set in the 'Text Color' control. On output, the final value in the 'Text Color' control. Pass 0 to hide the 'Text Color' control.</td>
</tr>
<tr>
<td>fontSize</td>
<td>integer</td>
<td>On input, the value to initially set in the 'Font Size' control. On output, the final value in the 'Font Size' control. Pass 0 to hide the 'Font Size' control.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
<td>Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>

Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User canceled out of dialog box</td>
</tr>
<tr>
<td>1</td>
<td>User modified the settings</td>
</tr>
</tbody>
</table>

Parameter Discussion

The valid values for justification are

VAL_LEFT_JUSTIFIED
VAL_RIGHT_JUSTIFIED
**GenericMessagePopup**

```c
int status = GenericMessagePopup (char title[], char message[], char buttonLabel1[], char buttonLabel2[], char buttonLabel3[], char responseBuffer[], int maxResponseLength, int buttonAlignment, int activeControl, int enterButton, int escapeButton);
```

**Purpose**

This function displays a dialog box with a user-defined message and optionally accepts a response string.

Up to three buttons are provided for generic use. The function returns a value indicating which button was pressed.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>title</th>
<th>string</th>
<th>The title of the dialog box.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>message</td>
<td>string</td>
<td>The message displayed on the dialog box.</td>
</tr>
<tr>
<td></td>
<td>buttonLabel1</td>
<td>string</td>
<td>The label on button 1.</td>
</tr>
<tr>
<td></td>
<td>buttonLabel2</td>
<td>string</td>
<td>The label on button 2. To hide buttons 2 and 3, pass 0 to Button Label 2.</td>
</tr>
<tr>
<td></td>
<td>buttonLabel3</td>
<td>string</td>
<td>The label on button 3. To hide button 3, pass 0 to Button Label 3.</td>
</tr>
<tr>
<td></td>
<td>maxResponseLength</td>
<td>integer</td>
<td>The maximum number of bytes the user is allowed to enter. The ResponseBuffer must be large enough to contain all of the user’s input plus one ASCII NULL byte.</td>
</tr>
<tr>
<td></td>
<td>buttonAlignment</td>
<td>integer</td>
<td>Selects the location of the three buttons for the generic message pop-up. A non-zero value aligns the buttons along the side of the dialog box. A value of zero aligns the buttons along the bottom of the dialog box.</td>
</tr>
<tr>
<td></td>
<td>activeControl</td>
<td>integer</td>
<td>Selects one of the three buttons or the input string as the active control. The active control is the control that accepts keystrokes when the dialog box is displayed. The following attribute names apply to activeControl. VAL_GENERIC_POPUP_BTN1 VAL_GENERIC_POPUP_BTN2 VAL_GENERIC_POPUP_BTN3 VAL_GENERIC_POPUP_INPUT_STRING</td>
</tr>
<tr>
<td></td>
<td>enterButton</td>
<td>integer</td>
<td>Selects which button has &lt;Enter&gt; as its shortcut key. If no button is to have &lt;Enter&gt; as its shortcut key, pass VAL_GENERIC_POPUP_NO_CTRL.</td>
</tr>
<tr>
<td></td>
<td>escapeButton</td>
<td>integer</td>
<td>Selects which button has &lt;Esc&gt; as its shortcut key. If no button is to have &lt;Esc&gt; as its shortcut key, pass VAL GENERIC POPUP NO_CTRL.</td>
</tr>
</tbody>
</table>

| Output   | responseBuffer | string  | The buffer in which the user’s response is stored. The buffer must be large enough to hold Max Response Length bytes plus one ASCII NULL byte. To hide the input string, pass 0 to Response Buffer. |
Return Value

| status    | integer | Refer to Appendix A for error codes. |

Return Codes

| 0          | VAL_GENERIC_POPUP_NO_CTRL |
| 1          | VAL_GENERIC_POPUP_BTN1    |
| 2          | VAL_GENERIC_POPUP_BTN2    |
| 3          | VAL_GENERIC_POPUP_BTN3    |

Get3dBorderColors

```c
int status = Get3dBorderColors(int baseColor, int *highlightColor,
                                int *lightColor, int *shadowColor,
                                int *darkShadowColor);
```

Purpose

Takes an RGB value for the base color of an object and returns the RGB values for colors that can be used to make the object look 3-dimensional. The colors returned are similar to the colors used in Windows 95 for drawing 3-dimensional objects.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>baseColor</th>
<th>integer</th>
<th>The RGB value for the color of an object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>highlightColor</td>
<td>integer</td>
<td>The RGB value for the color used to indicate the edges of the object that are in the most direct light.</td>
</tr>
<tr>
<td></td>
<td>lightColor</td>
<td>integer</td>
<td>The RGB value for the color used to indicate the transition between the highlight color and the base color of the object.</td>
</tr>
<tr>
<td></td>
<td>shadowColor</td>
<td>integer</td>
<td>The RGB value for the color used to indicate the edges of the object that are angled away from the light.</td>
</tr>
<tr>
<td></td>
<td>darkShadowColor</td>
<td>integer</td>
<td>The RGB value for the color used to indicate the edges of the object that are angled farthest away from the light.</td>
</tr>
</tbody>
</table>
Parameter Discussion

You may pass NULL for any of the output parameters.

Currently, the `lightColor` is always the same as the `baseColor`, as is the case in Windows 95.

Currently, the `darkShadowColor` is always black, as is the case in Windows 95.

---

**GetActiveCtrl**

```c
int activeCtrl = GetActiveCtrl(int panelHandle);
```

**Purpose**

This function obtains the ID of the active control on the specified panel.

The active control is the control that receives keyboard events when its panel is the active panel.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>activeCtrl</th>
<th>integer</th>
<th>Returns the control ID of the active control. The control ID was assigned to the control in the User Interface Editor, or returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function. Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

---

**GetActiveGraphCursor**

```c
int status = GetActiveGraphCursor(int panelHandle, int controlID, int *activeCursorNumber);
```

**Purpose**

Obtains the active cursor on the specified graph control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

Output

| Output     | activeCursorNumber | integer | Returns the number of the active cursor. The value will range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute. |

Return Value

| status     | integer           | Refer to Appendix A for error codes. |

GetActivePanel

```c
int activePanel = GetActivePanel (void);
```

Purpose

Obtains the handle of the active panel. The active panel is the panel that receives keyboard events.

Return Value

| activePanel | integer | Returns the handle of the active panel. This handle will have originally been assigned by LoadPanel, NewPanel, or DuplicatePanel. Refer to Appendix A for error codes. |
GetAxisItem

```c
int status = GetAxisItem(int panelHandle, int controlID, int axis,
                          int itemIndex, char itemLabel[], double *itemValue);
```

Purpose

This function retrieves the string/value pair at the specified index in the list of label strings for a graph or strip chart axis.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>axis</td>
<td>integer</td>
<td></td>
<td>Specifies the axis from which to retrieve the selected string/value pair. Valid values: VAL_XAXIS, VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS (graphs only).</td>
</tr>
<tr>
<td>itemIndex</td>
<td>integer</td>
<td></td>
<td>A zero-based index into the list of label strings.</td>
</tr>
<tr>
<td>Output</td>
<td>itemLabel</td>
<td>string</td>
<td>Buffer in which the label string is returned.</td>
</tr>
<tr>
<td></td>
<td>itemValue</td>
<td>double-precision</td>
<td>The value associated with the label string.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

Parameter Discussion

itemLabel must be large enough to hold the label string, including the terminating NUL byte. You can use GetAxisItemLabelLength to determine the length of the label string.

You may pass NULL for either of the output parameters.
GetAxisItemLabelLength

int status = GetAxisItemLabelLength(int panelHandle, int controlID, int axis, int itemIndex, int *length);

Purpose

This function obtains the number of characters in a label string for a graph or strip chart axis. The label string is specified by its index in the list of string/value pairs for that axis.

The length returned does not include the terminating NUL byte.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>axis</td>
<td>integer</td>
<td>Specifies the axis for which to return the length of the selected label string. Valid values: VAL_XAXIS VAL_LEFT_YAXIS VAL_RIGHT_YAXIS (graphs only)</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>A zero-based index into the list of label strings.</td>
</tr>
</tbody>
</table>

| Output  | length      | string  | The length of the selected label string. Excludes the terminating NUL byte. |

Return Value

| status  | integer | Refer to Appendix A for error codes. |
Parameter Discussion

**itemLabel** must be large enough to hold the label string, including the terminating NUL byte. You can use GetAxisItemLabelLength to determine the length of the label string.

You may pass NULL for either of the output parameters.

**See Also**

InsertAxisItem, GetNumAxisItems, GetAxisItem.

---

**GetAxisRange**

```c
int status = GetAxisRange(int panelHandle, int controlID, int *xAxisScalingMode, double *xmin, double *xmax, int *yAxisScalingMode, double *ymin, double *ymax);
```

**Purpose**

Obtains the scaling mode and the range of the X and Y axes for a graph or strip chart control.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

(continues)
Parameters (Continued)

| Output | xAxisScalingMode | integer | The current scaling mode of the X axis.  
|        |                 |        | 0L = VAL_MANUAL  
|        |                 |        | 1L = VAL_AUTOSCALE  
| xmin   | double-precision |        | The current minimum range of the x-axis.  
| xmax   | double-precision |        | The current maximum range of the x-axis.  
| yAxisScalingMode | integer | The current scaling mode of the Y axis.  
|        |                 |        | 0L = VAL_MANUAL  
|        |                 |        | 1L = VAL_AUTOSCALE.  
| ymin   | double-precision |        | The current minimum range of the Y axis.  
| ymax   | double-precision |        | The current maximum range of the Y axis.  

Return Value

| status | integer | Refer to Appendix A for error codes.  

Parameter Discussion

xAxisScalingMode

VAL_MANUAL The X axis is set to manual scaling, and its range is defined by Xmin and Xmax.

VAL_AUTOSCALE The X axis is set to auto scaling. Xmin and Xmax are not returned. Auto scaling is not allowed for strip charts.

yAxisScalingMode

VAL_MANUAL The Y axis is set to manual scaling, and its range is defined by Ymin and Ymax.

VAL_AUTOSCALE The Y axis is set to auto scaling. Ymin and Ymax are not returned. Auto scaling is not allowed for strip charts.
GetAxisScalingMode

```c
int status = GetAxisScalingMode(int panelHandle, int controlID, int axis,
                                 int *axisScaling, double *min, double *max);
```

Purpose

Obtains the scaling mode and the range of any graph axis or the Y axis of a strip chart.

This function is not valid for the X axis of a strip chart. To obtain the X offset and X increment for a strip chart, use the `GetCtrlAttribute` function with the `ATTR_XAXIS_OFFSET` and `ATTR_XAXIS_GAIN` attributes.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>axis</td>
<td>integer</td>
<td>Specifies for which axis to obtain the mode and range. Valid values: <code>VAL_XAXIS</code> (graphs only) <code>VAL_LEFT_YAXIS</code> (graphs and strip charts) <code>VAL_RIGHT_YAXIS</code> (graphs only)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>axisScaling</th>
<th>integer</th>
<th>The scaling mode used for the axis. See table below.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>double-precision</td>
<td>The current minimum value on the axis.</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>double-precision</td>
<td>The current maximum value on the axis.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>
Parameter Discussion

axisScaling is one of the following values.

<table>
<thead>
<tr>
<th>Valid Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_MANUAL</td>
<td>The axis is set to manual scaling, and its range is defined by ( \text{min} ) and ( \text{max} ).</td>
</tr>
<tr>
<td>VAL_AUTOSCALE</td>
<td>The axis is set to auto scaling. ( \text{min} ) and ( \text{max} ) are not used. Auto scaling is not valid for strip charts.</td>
</tr>
<tr>
<td>VAL_LOCK</td>
<td>The axis is set to manual scaling using the current (possibly auto scaled) minimum and maximum values on the axis. VAL_LOCK is not valid for strip charts.</td>
</tr>
</tbody>
</table>

If you call SetAxisScalingMode with axisScaling set to VAL_AUTOSCALE, and you then call SetAxisScalingMode with axisScaling set to VAL_LOCK, GetAxisScalingMode returns axisScaling as VAL_MANUAL.

max always exceeds min.

You may pass NULL for any of the output parameters.

See Also

SetAxisScalingMode

GetBitmapData

```c
int status = GetBitmapData(int bitmapID, int *bytesPerRow, int *pixelDepth, int *width, int *height, int colorTable[], unsigned char bits[], unsigned char mask[]);
```

Purpose

Obtains the bit values that define the image associated with a bitmap. Before calling GetBitmapData, you must do one of the following.

- Call GetBitmapInfo to get the size of the buffers needed, and then allocate the buffers, or
- Call AllocBitmapData.
### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>bitmapID</th>
<th>integer</th>
<th>The ID of the bitmap object containing the image. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, or GetPanelDisplayBitmap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>bytesPerRow</td>
<td>integer</td>
<td>The number of bytes on each scan line of the image.</td>
</tr>
<tr>
<td></td>
<td>pixelDepth</td>
<td>integer</td>
<td>The number of bits per pixel.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>The width of the image, in pixels.</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>integer</td>
<td>The height of the image, in pixels.</td>
</tr>
<tr>
<td></td>
<td>colorTable</td>
<td>integer array</td>
<td>An array of RGB color values, or NULL if pixelDepth is greater than 8.</td>
</tr>
<tr>
<td></td>
<td>bits</td>
<td>unsigned char array</td>
<td>An array of bits that determine the colors to be displayed on each pixel in the image.</td>
</tr>
<tr>
<td></td>
<td>mask</td>
<td>unsigned char array</td>
<td>An array containing one bit per pixel in the image. Each bit specifies whether the pixel is actually drawn. May be NULL.</td>
</tr>
</tbody>
</table>

### Return Value

| status | integer | Refer to Appendix A for error codes. |

### Parameter Discussion

If there is no image, the `width` and `height` parameters are set to -1. If the bitmap originated from a Windows metafile (.WMF), the size of the bitmap obtained by this function is the size stored in the original Windows metafile.

The `pixelDepth` parameter is set to 1, 2, 8, 24, or 32.

The number of bits in the `bits` array for each pixel is equal to the `pixelDepth` value. If `pixelDepth` is 8 or less, the `bits` array is filled with indices into the `colorTable` array, and the number of entries in the `colorTable` array is 2 raised to the power of the `pixelDepth` parameter. If `pixelDepth` is greater than 8, the `colorTable` array is not used, and the `bits` array contains the actual RGB values.

For a `pixelDepth` of 24, each pixel is represented by a 3-byte RGB value of the form 0xRRGGBB, where RR, GG, and BB represent the red, green and blue intensity of the color. The RR byte is always at the lowest memory address of the three bytes.
If the `pixelDepth` is 32, each pixel in the `bits` array is represented by a 32-bit RGB value of the form `0x00RRGGBB`, where `RR`, `GG`, and `BB` represent the red, green and blue intensity of the color. The 32-bit value is treated as a native 32-bit integer value for the platform. The most significant byte is always ignored. The `BB` byte should always be in the least significant byte. On a little-endian platform (for example, Intel processors), `BB` would be at the lowest memory address. On a big-endian platform (for example, Motorola processors), `BB` would be at the highest address. Note that this differs from the format of the `bits` array when the `pixelDepth` is 24.

The first pixel in the `bits` array is at the top, left corner of the image. The pixels in the array are in row-major order.

If `GetBitmapInfo` sets the `colorSize` parameter to zero, the `colorTable` array is not filled in.

In the `mask` array, a bit value of 1 indicates that the pixel is drawn. 0 indicates that the pixel is not drawn. Exception: If an image has a `pixelDepth` of 1, pixels with a `bits` value of 1 (foreground pixels) are always drawn and the `mask` affects only the pixels with a bitmap of 0 (background pixels). Each row of the mask is padded to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then there are 32 bits (in other words, 4 bytes) of data in each row of the mask.

A mask is useful for achieving transparency.

If `GetBitmapInfo` sets the `maskSize` parameter to zero, the `mask` array is not filled in.

You may pass NULL for any of the output parameters.

### See Also


---

### `GetBitmapFromFile`

```c
int status = GetBitmapFromFile(char fileName[], int *bitmapID);
```

### Purpose

Reads a bitmap image from a file and creates a bitmap object. The bitmap ID can then be passed to any function that accepts a bitmap, such as `CanvasDrawBitmap` or `ClipboardPutBitmap`.

You can discard the bitmap object by passing its ID to `DiscardBitmap`.

You can use the following image types:

- **PCX**: Windows and SPARCstation
BMP, DIB, RLE, ICO: Windows only
WMF: Windows 95 and NT only
XWD: SPARCstation only

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>fileName</th>
<th>string</th>
<th>The pathname of the file which contains the image. If the name is a simple filename, the file is loaded from the project. If it is not found in the project, the file is loaded from the directory containing the project.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>bitmapID</td>
<td>integer</td>
<td>An ID that serves as a handle to the bitmap object.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

ClipboardPutBitmap, GetBitmapData, SetCtrlBitmap, PlotBitmap, CanvasDrawBitmap, DiscardBitmap.

GetBitmapInfo

```c
int status = GetBitmapInfo (int bitmapID, int *colorSize, int *bitsSize, int *maskSize);
```

Purpose

Obtains size information about the image associated with a bitmap. This information can then be used in allocating the buffers to be passed to the GetBitmapData function.

As an alternative to this function, you can call AllocBitmapData, which allocates the buffers for you.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>bitmapID</th>
<th>integer</th>
<th>The ID of the bitmap object containing the image. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, or GetPanelDisplayBitmap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>colorSize</td>
<td>integer</td>
<td>The number of bytes in the image color table (0 if the pixel depth of the image is greater than 8).</td>
</tr>
<tr>
<td></td>
<td>bitsSize</td>
<td>integer</td>
<td>The number of bytes in the image bitmap.</td>
</tr>
<tr>
<td></td>
<td>maskSize</td>
<td>integer</td>
<td>The number of bytes in the image mask (0 if there is no mask).</td>
</tr>
</tbody>
</table>

Return Value

| status  | integer | Refer to Appendix A for error codes. |

Parameter Discussion

You may pass NULL for any of the output parameters.

See Also

GetBitmapData, AllocBitmapData.

GetCtrlAttribute

```c
int status = GetCtrlAttribute(int panelHandle, int controlID, int controlAttribute, void *attributeValue);
```

Purpose

Obtains the value of a control attribute from the selected panel and control.

Since attributes may Return Values of different data types with different valid ranges, a list of attributes, their data types and valid values are provided in Table 3-9 in Chapter 3, Programming with the User Interface Library.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>controlAttribute</td>
<td>integer</td>
<td>Selects a control attribute. Many control attributes are specific to one kind of control, while others are used with all or many different controls. See Table 3-9 in Chapter 3 for a complete listing of control attributes.</td>
</tr>
<tr>
<td>Output</td>
<td>attributeValue</td>
<td>void *</td>
<td>Returns the value of the specified control attribute. See Table 3-9 in Chapter 3 for a complete listing of control attributes.</td>
</tr>
</tbody>
</table>

Return Value

| status     | integer | Refer to Appendix A for error codes. |

GetCtrlBitmap

```c
int status = GetCtrlBitmap(int panelHandle, int controlID, int imageID, int *bitmapID);
```

Purpose

Obtains a bitmap image from a control and stores it in a bitmap object. The bitmap ID can then be passed to any function that accepts a bitmap, such as CanvasDrawBitmap or ClipboardPutBitmap.

The following control types can contain images:

- picture controls
- picture rings
- picture buttons
- graph controls
- canvas controls
You can use this function on images set using DisplayImageFile, InsertListItem, ReplaceListItem, PlotBitmap, or SetImageBits, or SetCtrlAttribute with the ATTR_IMAGE_FILE attribute.

You can discard the bitmap object by passing its ID to DiscardBitmap.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>imageID</td>
<td>integer</td>
<td>For picture rings, the zero-based index of an image in the ring. For graphs, this argument is the plotHandle returned from PlotBitmap. For picture controls, picture buttons, and canvas controls, this argument is ignored.</td>
</tr>
<tr>
<td>Output</td>
<td>bitmapID</td>
<td>integer</td>
<td>An ID that serves as a handle to the bitmap object.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

**See Also**

ClipboardPutBitmap, GetBitmapData, SetCtrlBitmap, PlotBitmap, CanvasDrawBitmap, DiscardBitmap.

---

**GetCtrlBoundingRect**

```c
int status = GetCtrlBoundingRect(int panelHandle, int controlID, int *top, int *left, int *height, int *width);
```

**Purpose**

This function returns the top, left, width, and height coordinates for the control’s bounding rectangle.

The control’s labels are included within the bounding rectangle.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>top</th>
<th>integer</th>
<th>Returns the top coordinate of the control’s bounding rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>left</td>
<td>integer</td>
<td>Returns the left coordinate of the control’s bounding rectangle.</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>integer</td>
<td>Returns the vertical size of the control’s bounding rectangle.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>Returns the horizontal size of the control’s bounding rectangle.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

Parameter Discussion

You can pass 0 (zero) to any parameter from which you are not interested in getting the value. The range of the top and left Parameters is -32768 to 32767. The range of the height and width Parameters is 1 to 32767. The origin (0,0) is at the upper left corner of the panel (before the panel is scrolled) directly below the title bar.

GetCtrlDisplayBitmap

```c
int status = GetCtrlDisplayBitmap(int panelHandle, int controlID,
                                   int includeLabel, int *bitmapID);
```

Purpose

This function creates a bitmap object containing a "snapshot" image of the current appearance of the specified control. The bitmap ID can then be passed to any function that accepts a bitmap, such as CanvasDrawBitmap or ClipboardPutBitmap.
For example, you can paste a picture of a control onto the system clipboard by calling `GetCtrlDisplayBitmap` and then passing the bitmap ID to `ClipboardPutBitmap`.

You can discard the bitmap object by passing the ID to the `DiscardBitmap` function.

### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>includeLabel</td>
<td>integer</td>
<td>If nonzero, the control label is included in the image.</td>
</tr>
<tr>
<td>Output</td>
<td>bitmapID</td>
<td>integer</td>
<td>An ID that serves as a handle to the bitmap object.</td>
</tr>
</tbody>
</table>

### Return Value

| status         | integer | Refer to Appendix A for error codes. |

### See Also


---

### GetCtrlIndex

```c
int status = GetCtrlIndex(int panelHandle, int controlID, int *itemIndex);
```

**Purpose**

This function returns the current index of the specified list control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

| Output  | itemIndex   | integer | Returns the current index of the specified list control. Returns -1 if the list control has no items. |

Return Value

| status  | integer     | Refer to Appendix A for error codes. |

GetCtrlVal

```c
int status = GetCtrlVal (int panelHandle, int controlID, void *value);
```

Purpose

Obtains the current value of a control.

When called on a list control (a list box or ring), GetCtrlVal returns the value of the current (selected) list item. To obtain the index of the selected list item, use the GetCtrlIndex function.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

| Output  | value       | void *  | Returns the control’s value. The data type of value must match the data type of the control. |


Return Value

| status       | integer | Refer to Appendix A for error codes. |

GetCursorAttribute

```c
int status = GetCursorAttribute(int panelHandle, int controlID, int cursorNumber, int cursorAttribute, int *attributeValue);
```

Purpose

Obtains one of the following graph cursor attributes: mode, point style, cross hair style, color, y-axis.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>cursorNumber</td>
<td>integer</td>
<td>The cursor number may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
</tr>
<tr>
<td></td>
<td>cursorAttribute</td>
<td>integer</td>
<td>Selects a particular graph cursor attribute. Valid Attributes: ATTR_CURSOR_MODE, ATTR_CURSOR_POINT_STYLE, ATTR_CROSS_HAIR_STYLE, ATTR_CURSOR_COLOR, ATTR_CURSOR_YAXIS</td>
</tr>
</tbody>
</table>

| Output      | attributeValue | integer | See Table 3-20 in Chapter 3 for a complete listing of cursor attributes. |

Return Value

| status       | integer | Refer to Appendix A for error codes. |
GetCVITaskHandle

```c
int taskHandle = GetCVITaskHandle(void);
```

**Note:** *This function is available only on the Windows 3.1 version of LabWindows/CVI.*

**Purpose**

This function returns the Windows 3.1 task handle associated with the LabWindows/CVI application. This number can be used by a DLL requiring LabWindows/CVI’s Windows task handle.

For Windows 95 and NT, use the Windows API function `GetCurrentProcessID`.

**Return Value**

| taskHandle | integer | The Windows task handle associated with the LabWindows/CVI application. |

---

GetCVIWindowHandle

```c
int windowHandle = GetCVIWindowHandle(void);
```

**Note:** *This function is available only on the Windows version of LabWindows/CVI.*

**Purpose**

This function returns the window handle associated with the LabWindows/CVI application. This number should be used in the DLL as the hwnd parameter for the Windows function, `PostMessage`, as indicated in the Windows SDK documentation.

To make this work, you call `RegisterWinMsgCallback` and `GetCVIWindowHandle` and then pass their Return Values (uMsg and hwnd) to the DLL. When the DLL wants to send a message, it calls `PostMessage` with these values. When LabWindows/CVI receives the message, it calls the callback function.

**Note:** *LabWindows/CVI can receive the message only when it is processing events. LabWindows/CVI processes events when it is waiting for user input. If the program running in LabWindows/CVI does not call `RunUserInterface`, `GetUserEvent`, or `scanf`, or if it does not return from a User Interface Library callback, events will not be processed. This can be remedied in the program by calling the User Interface Library function `ProcessSystemEvents` periodically.*
GetGlobalMouseState

```c
int status = GetGlobalMouseState(int *panelHandle, int *xCoordinate,
                                 int *yCoordinate, int *leftButtonDown,
                                 int *rightButtonDown, int *keyModifiers);
```

**Purpose**

Obtains information about the state of the mouse cursor. `xCoordinate` and `yCoordinate` are set to the position of the mouse relative to the top, left corner of the screen.

**Parameters**

<table>
<thead>
<tr>
<th>Output</th>
<th>panelHandle</th>
<th>integer</th>
<th>The handle of the panel that the mouse is over. 0 if the mouse is not over a panel.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xCoordinate</td>
<td>integer</td>
<td>The x-coordinate of the mouse cursor relative to the left edge of the screen.</td>
</tr>
<tr>
<td></td>
<td>yCoordinate</td>
<td>integer</td>
<td>The y-coordinate of the mouse cursor relative to the top of the screen.</td>
</tr>
<tr>
<td></td>
<td>leftButtonDown</td>
<td>integer</td>
<td>0 if the left button is up. 1 if the left button is down.</td>
</tr>
<tr>
<td></td>
<td>rightButtonDown</td>
<td>integer</td>
<td>0 if the right button is up. 1 if the right button is down.</td>
</tr>
<tr>
<td></td>
<td>keyModifiers</td>
<td>integer</td>
<td>The state of the keyboard modifiers the (in other words, the &lt;Ctrl&gt; and &lt;Shift&gt; keys). If no keys are down, the value is 0. Otherwise, the value is the bitwise 'OR' of the appropriate key masks: VAL_MENUKEY_MODIFIER and VAL_SHIFT_MODIFIER.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

`windowHandle` integer The Windows handle associated with the LabWindows/CVI application.
Parameter Discussion

You may pass NULL (0) in place of any of the output parameters.

See Also

GetRelativeMouseState.

GetGraphCursor

```c
int status = GetGraphCursor (int panelHandle, int controlID, int cursorNumber, 
    double *x, double *y);
```

Purpose

Obtains the current position of the specified graph cursor.

The position is relative to the current range of the X and Y axes.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>cursorNumber</td>
<td>integer</td>
<td>The cursor number may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
</tr>
<tr>
<td>Output</td>
<td>x</td>
<td>double-precision</td>
<td>Returns the X coordinate of the cursor position.</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>double-precision</td>
<td>Returns the Y coordinate of the cursor position.</td>
</tr>
</tbody>
</table>
Return Value

| status       | integer | Refer to Appendix A for error codes. |

GetGraphCursorIndex

```c
int status = GetGraphCursorIndex (int panelHandle, int controlID, int cursorNumber,
int *plotHandle, int *arrayIndex);
```

Purpose

Obtains the plot handle and array index of the plot on which the cursor is attached.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>cursorNumber</td>
<td>integer</td>
<td>The cursor number may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>plotHandle</th>
<th>integer</th>
<th>Contains the ID of the plot to which the cursor is attached. If the cursor is not attached to a plot, -1 is returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>arrayIndex</td>
<td>integer</td>
<td>Contains the array index of the data point to which the cursor is attached. If the cursor is attached to a point plot, or if the cursor is not attached to a data point, -1 is returned.</td>
</tr>
</tbody>
</table>

Return Value

| status       | integer | Refer to Appendix A for error codes. |
GetImageBits

```c
int status = GetImageBits (int panelHandle, int controlID, int imageID,
    int *rowBytes, int *depth, int *width, int *height,
    int colorTable[], unsigned char bitmap[],
    unsigned char mask[]);
```

**Purpose**

Obtains the bit values that define an image. You can modify the bit values and then apply them back to the image using `setImageBits`.

Before calling `GetImageBits`,

- Call `GetImageInfo` to get the size of the buffers needed, and then allocate the buffers, or
- Call `AllocImageBits`.

The following control types can contain images.

- picture controls
- picture rings
- picture buttons
- graph controls

This function can be used in images set via `DisplayImageFile`, `InsertListItem`, `ReplaceListItem`, `PlotBitmap`, or `setImageBits`, or `SetCtrlAttribute` with the `ATTR_IMAGE_FILE` attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the user Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>imageID</td>
<td>integer</td>
<td>For a picture ring, the zero-based index of an image in the ring. For a graph, the <code>plotHandle</code> returned from <code>PlotBitmap</code>. For picture controls and buttons, this is ignored.</td>
</tr>
</tbody>
</table>
(Continued)

<table>
<thead>
<tr>
<th>Output</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowBytes</td>
<td>integer</td>
<td>The number of bytes on each scan line of the image.</td>
</tr>
<tr>
<td>depth</td>
<td>integer</td>
<td>The number of bits per pixel.</td>
</tr>
<tr>
<td>width</td>
<td>integer</td>
<td>The width of the image, in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>integer</td>
<td>The height of the image, in pixels.</td>
</tr>
<tr>
<td>colorTable</td>
<td>integer array</td>
<td>An array of RGB color values.</td>
</tr>
<tr>
<td>bitmap</td>
<td>unsigned char array</td>
<td>An array of bits that determine the colors to be displayed on each pixel in the image.</td>
</tr>
<tr>
<td>mask</td>
<td>unsigned char array</td>
<td>An array containing one bit per pixel in the image. Each bit specifies whether the pixel is actually drawn.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

Parameter Discussion

If there is no image, the **width** and **height** parameters are set to -1. If the image was originally rendered from a Windows metafile (.WMF), the size of the bitmap obtained by this function is determined as follows.

- If the image was on a graph control or if the fit mode is **VAL_SIZE_TO_PICTURE**, the size of the bitmap is the size of the image as it appears in the control.

- If the image is on a picture, picture ring, or picture button, and the fit mode is other than **VAL_SIZE_TO_PICTURE**, the size of the bitmap is the size stored in the original Windows metafile.

The number of entries in the **colorTable** array must be equal to 2 raised to the power of the **depth** parameter.

The **pixelDepth** parameter is set to 1, 2, 8, 24, or 32.

The number of bits in the **bits** array for each pixel is equal to the **pixelDepth** value. If **pixelDepth** is 8 or less, the **bits** array is filled with indices into the **colorTable** array, and the number of entries in the **colorTable** array is 2 raised to the power of the **pixelDepth** parameter. If **pixelDepth** is greater than 8, the **colorTable** array is not used, and the **bits** array contains the actual RGB values.
For a `pixelDepth` of 24, each pixel is represented by a 3-byte RGB value of the form $0xRRGGBB$, where RR, GG, and BB represent the red, green and blue intensity of the color. The RR byte is always at the lowest memory address of the three bytes.

If the `pixelDepth` is 32, each pixel in the `bits` array is represented by a 32-bit RGB value of the form $0x00RRGGBB$, where RR, GG, and BB represent the red, green and blue intensity of the color. The 32-bit value is treated as a native 32-bit integer value for the platform. The most significant byte is always ignored. The BB byte should always be in the least significant byte. On a little-endian platform (for example, Intel processors), BB would be at the lowest memory address. On a big-endian platform (for example, Motorola processors), BB would be at the highest address. Note that this differs from the format of the bits array when the `pixelDepth` is 24.

The first pixel in the `bits` array is at the top, left corner of the image. The pixels in the array are in row-major order.

If `GetBitmapInfo` sets the `colorSize` parameter to zero, the `colorTable` array is not filled in.

In the `mask` array, a bit value of 1 indicates that the pixel is drawn. 0 indicates that the pixel is not drawn. Exception: If an image has a `pixelDepth` of 1, pixels with a `bits` value of 1 (foreground pixels) are always drawn and the `mask` affects only the pixels with a bitmap of 0 (background pixels). Each row of the mask is padded to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then there are 32 bits (in other words, 4 bytes) of data in each row of the mask.

A mask is useful for achieving transparency.

If `GetImageInfo` sets the `maskSize` parameter to zero, the `mask` array will not be filled in.

You may pass NULL for any of the output parameters.

You can now use a 24-bit pixel depth in the four picture control image bits functions. When you do, the color table parameters to the functions are not used. The array of bits contains RGB values rather than indexes into the color table. Each RGB value in the array of bits represented by a 3-byte value of the form $0xRRGGBB$

where RR, GG, and BB represent the red, green and blue intensity of the color. The RR byte should always be at the lowest memory address of the three bytes.

**See Also**

`GetImageInfo`, `AllocImageBits`, `SetImageBits`.
GetImageInfo

```c
int status = GetImageInfo(int panelHandle, int controlID, int imageID,
                          int *colorSize, int *bitmapSize, int *maskSize);
```

**Purpose**

Obtains size information about an image associated with a control on a panel. This information can then be used in allocating the buffers to be passed to the `GetImageBits` function.

The following control types can contain images.

- picture controls
- picture rings
- picture buttons
- graph controls

As an alternative to this function, you can call `AllocImageBits`, which allocates the buffers for you.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the user Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>imageID</td>
<td>integer</td>
<td>For a picture ring, the zero-based index of an image in the ring. For a graph, the <code>plotHandle</code> returned from <code>PlotBitmap</code>. For picture controls and buttons, this is ignored.</td>
</tr>
<tr>
<td>Output</td>
<td>colorSize</td>
<td>integer</td>
<td>The number of bytes in the image color table (-1 if there is no image).</td>
</tr>
<tr>
<td></td>
<td>bitmapSize</td>
<td>integer</td>
<td>The number of bytes in the image bitmap (-1 if there is no image).</td>
</tr>
<tr>
<td></td>
<td>maskSize</td>
<td>integer</td>
<td>The number of bytes in the image mask (0 if the image has no mask, -1 if there is no image.)</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |
Parameter Discussion

You may pass NULL for any of the output parameters.

You can now use a 24-bit pixel depth in the four picture control image bits functions. When you do, the color table parameters to the functions are not used. The array of bits contains RGB values rather than indexes into the color table. Each RGB value in the array of bits represented by a 3-byte value of the form

0xRRGGBB

where RR, GG, and BB represent the red, green and blue intensity of the color. The RR byte should always be at the lowest memory address of the three bytes.

See Also

GetImageBits.

---

GetIndexFromValue

int status = GetIndexFromValue (int panelHandle, int controlID, int *index, ...);

Purpose

This function returns the index of the first list item with the specified value.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemValue</td>
<td>depends on the type of the control</td>
<td>Specifies the value whose index will be returned.</td>
</tr>
</tbody>
</table>

| Output  | index       | integer | Returns the index of the first list item with the specified value. |


### GetLabelFromIndex

```c
int status = GetLabelFromIndex(int panelHandle, int controlID, int itemIndex,
                                 char itemLabel[]);
```

#### Purpose

This function returns the label of the specified list item.

For picture ring controls, this function returns an empty string.

#### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list.</td>
</tr>
</tbody>
</table>

| Output  | itemLabel   | string  | Returns the label of the selected list item. |

#### Return Value

| status | integer | Refer to Appendix A for error codes. |
GetLabelLengthFromIndex

```c
int status = GetLabelLengthFromIndex (int panelHandle, int controlID,
    int itemIndex, int *length);
```

**Purpose**

This function returns the label length of the specified list item.

For picture ring controls, this function returns zero.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list.</td>
</tr>
<tr>
<td>Output</td>
<td>length</td>
<td>integer</td>
<td>Returns the number of characters in the label of the selected list item.</td>
</tr>
</tbody>
</table>

**Return Value**

| status   | integer   | Refer to Appendix A for error codes. |

GetListItemImage

```c
int status = GetListItemImage (int panelHandle, int controlID, int itemIndex,
    int *image);
```

**Purpose**

This function returns a value corresponding to the image state of the specified list item.

The image state of a list item indicates the presence of various folder icons next to the item.

This function is not valid for picture ring controls. For picture rings, see GetImageBits.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list.</td>
</tr>
</tbody>
</table>

Output | image | integer | Returns the value corresponding to the image state of the specified list item. The possible image states are:  
0 = VAL_NO_IMAGE  
1 = VAL_FOLDER  
2 = VAL_OPEN_FOLDER  
3 = VAL_CURRENT_FOLDER |

Return Value

| status | integer | Refer to Appendix A for error codes. |

GetMenuBarAttribute

```c
int status = GetMenuBarAttribute(int menuBarHandle, int menuorMenuItemID, int menuBarAttribute, void *attributeValue);
```

Purpose

This function returns the value of the specified menu bar attribute.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The ID which was assigned to the menu or menu item by the User Interface Editor (located in the .uir header file), or returned by NewMenu, NewSubMenu, or NewMenuItem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuorMenuItemID</td>
<td>integer</td>
<td></td>
<td>The ID which was assigned to the menu or menu item by the User Interface Editor (located in the .uir header file), or returned by NewMenu, NewSubMenu, or NewMenuItem. If the attribute corresponds to the entire menu bar, pass 0 (zero) as this parameter.</td>
</tr>
<tr>
<td>menuBarAttribute</td>
<td>integer</td>
<td></td>
<td>A particular menu bar attribute. See Table 3-6 in Chapter 3 for a complete listing of menu bar attributes.</td>
</tr>
</tbody>
</table>

| Output         | attributeValue | void * | The value of the selected menu bar attribute. See Table 3-6 in Chapter 3 for valid values associated with all menu bar attributes. |

Return Value

| status         | integer       | Refer to Appendix A for error codes. |

GetMouseCursor

```c
int status = GetMouseCursor (int *mouseCursorStyle);
```

Purpose

Returns the mouse cursor style set by SetMouseCursor.

Parameters

| Output         | mouseCursorStyle | integer | Returns the mouse cursor style set by SetMouseCursor. The possible mouse cursor styles are shown in Table 3-4 of Chapter 3. |

Return Value

| status         | integer       | Refer to Appendix A for error codes. |
GetNumAxisItems

```c
int status = GetNumAxisItems(int panelHandle, int controlID, int axis,
                               int *count);
```

Purpose

Returns the number of items in the list of label strings for a graph or strip chart axis.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>axis</td>
<td>integer</td>
<td>Specifies the axis for which to return the count of string/value pairs. Valid values: VAL_XAXIS, VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS (graphs only)</td>
</tr>
</tbody>
</table>

| Output | count       | integer       | The number of string/value pairs for the axis. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

InsertAxisItem, GetAxisItem, GetAxisItemLabelLength
GetNumCheckedItems

```
int status = GetNumCheckedItems (int panelHandle, int controlID,
   int *numberOfItems);
```

**Purpose**

This function returns the number of checked list items from a specified list control.

This function only applies to list boxes with the check mode attribute set.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

| Output  | numberOfItems | integer | Returns the number of list items that are checked. |

**Return Value**

| status  | integer | Refer to Appendix A for error codes. |

---

GetNumListItems

```
int status = GetNumListItems (int panelHandle, int controlID, int *count);
```

**Purpose**

This function returns the number of list items in the specified list control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

Output | count | integer | Returns the number of items in the specified list control. Because the indices are zero-based, this value will be 1 greater than the index of the last item. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

GetNumTextBoxLines

```
int status = GetNumTextBoxLines(int panelHandle, int controlID, int *count);
```

Purpose

This function returns the number of lines containing text in the specified text box.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
</tbody>
</table>

Output | count | integer | Returns the number of lines containing text in the specified text box. |
GetPanelAttribute

```c
int status = GetPanelAttribute(int panelHandle, int panelAttribute,
    void *attributeValue);
```

**Purpose**

This function returns the value of a particular panel attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelAttribute</td>
<td>integer</td>
</tr>
</tbody>
</table>

- The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.
- See Table 3-2 in Chapter 3 for a complete listing of panel attributes.

| Output      | attributeValue  | void *        |

- The value of the specified panel attribute. See Table 3-2 in Chapter 3 for valid values associated with all panel attributes.

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

GetPanelDisplayBitmap

```c
int status = GetPanelDisplayBitmap(int panelHandle, int scope, Rect area,
    int *bitmapID);
```

**Purpose**

This function creates a bitmap object containing a "snapshot" image of the current appearance of the specified panel. The bitmap ID can then be passed any function that accepts a bitmap, such as CanvasDrawBitmap or ClipboardPutBitmap.
For example, you can paste a picture of a panel onto the system clipboard by calling GetPanelDisplayBitmap and then passing the bitmap ID to ClipboardPutBitmap.

You can discard the bitmap object by passing the ID to the DiscardBitmap function.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td><strong>scope</strong></td>
<td>integer</td>
<td>Determines which portions of the specified panel are to be copied to the bitmap. See table of valid values below.</td>
</tr>
<tr>
<td></td>
<td><strong>area</strong></td>
<td>Rect</td>
<td>Use this parameter to restrict the area of the panel copied into the bitmap. The rectangle coordinates are relative to the upper-left corner of the panel (directly below the title bar) before the panel is scrolled. Use VAL_ENTIRE_OBJECT if you do not want to restrict the area copied.</td>
</tr>
</tbody>
</table>

| Output | **bitmapID** | integer | An ID that serves as a handle to the bitmap object. |

**Return Value**

| **status** | integer | Refer to Appendix A for error codes. |

The valid values for the scope parameter are:

<table>
<thead>
<tr>
<th><strong>Valid Values</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_VISIBLE_AREA</td>
<td>The visible area of the panel is copied to the bitmap, including the frame, menu bar, and scroll bars.</td>
</tr>
<tr>
<td>VAL_FULL_PANEL</td>
<td>The entire contents of the panel are copied to the bitmap, excluding the frame, menu bar, and scroll bars. This includes contents that might currently be scrolled off the visible area.</td>
</tr>
</tbody>
</table>
See Also

ClipboardPutBitmap, GetBitmapData, GetCtrlDisplayBitmap, SetCtrlBitmap, PlotBitmap, CanvasDrawBitmap, DiscardBitmap.

GetPanelMenuBar

```c
int menuBarHandle = GetPanelMenuBar(int panelHandle);
```

**Purpose**

This function returns the handle of the menu bar associated with the specified panel.

**Parameters**

| Input  | panelHandle | integer | The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. |

**Return Value**

| menuBarHandle | integer | The menu bar associated with the specified panel. The menu bar handle would have been returned by LoadMenuBar or NewMenuBar. If no menu bar is associated with the panel, zero is returned. Refer to Appendix A for error codes. |

GetPlotAttribute

```c
int status = GetPlotAttribute(int panelHandle, int controlID, int plotHandle, int plotAttribute, void *attributeValue);
```

**Purpose**

Obtains the value of a graph plot attribute.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the user Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for a particular plot in the graph. It will have been returned by one of the graph plotting functions.</td>
</tr>
<tr>
<td></td>
<td>plotAttribute</td>
<td>integer</td>
<td>Selects a particular graph plot attribute. See Table 3-20 in Chapter 3 for a complete listing of plot attributes.</td>
</tr>
<tr>
<td>Output</td>
<td>attributeValue</td>
<td>void *</td>
<td>Returns the value of the specified plot attribute. See Table 3-20 in Chapter 3 for a complete listing of plot attributes.</td>
</tr>
</tbody>
</table>

Return Value

| status    | integer | Refer to Appendix A for error codes. |

GetPrintAttribute

```c
int status = GetPrintAttribute(int printAttribute, int *attributeValue);
```

Purpose

Returns the value of the selected print attribute.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>printAttribute</th>
<th>integer</th>
<th>See Table 3-27 in Chapter 3 for a complete listing of print attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>attributeValue</td>
<td>integer</td>
<td>The current value of the print attribute. See Table 3-27 in Chapter 3 for valid values associated with all print attributes.</td>
</tr>
</tbody>
</table>
Return Value

| status  | integer | Refer to Appendix A for error codes. |

GetRelativeMouseState

```c
int status = GetRelativeMouseState(int panelHandle, int controlID,
   int *xCoordinate, int *yCoordinate,
   int *leftButtonDown, int *rightButtonDown,
   int *keyModifiers);
```

Purpose

Obtains information about the state of the mouse cursor. xCoordinate and yCoordinate are set to the position of the mouse relative to the top and left coordinates of the control specified. If controlID is 0, the coordinate information is relative to the top and left coordinates of the panel specified (excluding the panel frame and title bar). Note that the coordinates are returned even if the mouse is not over the control or the panel.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the user Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function. Pass 0 to indicate that the coordinates should be relative to the panel.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>xCoordinate</th>
<th>integer</th>
<th>The x-coordinate of the mouse cursor relative to the left edge of the control (or panel, if control ID is 0).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yCoordinate</td>
<td>integer</td>
<td>The y-coordinate of the mouse cursor relative to the top of the control (or panel, if controlID is 0).</td>
</tr>
<tr>
<td></td>
<td>leftButtonDown</td>
<td>integer</td>
<td>0 if the left button is up. 1 if the left button is down.</td>
</tr>
<tr>
<td></td>
<td>rightButtonDown</td>
<td>integer</td>
<td>0 if the right button is up. 1 if the right button is down.</td>
</tr>
<tr>
<td></td>
<td>keyModifiers</td>
<td>integer</td>
<td>The state of the keyboard modifiers the (in other words, the &lt;Ctrl&gt; and &lt;Shift&gt; keys). If no keys are down, the value is 0. Otherwise, the value is the bitwise ‘OR’ of the appropriate key masks: VAL_MENUKEY_MODIFIER and VAL_SHIFT_MODIFIER.</td>
</tr>
</tbody>
</table>
Return Value

| status   | integer | Refer to Appendix A for error codes. |

Parameter Discussion

You may pass NULL (0) in place of any of the output parameters.

See Also

GetGlobalMouseState.

GetScreenSize

```c
int status = GetScreenSize (int *height, int *width);
```

Purpose

Returns the height and width of the screen in pixels.

Parameters

<table>
<thead>
<tr>
<th>Output</th>
<th>height</th>
<th>integer</th>
<th>Height of the screen in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>Width of the screen in pixels.</td>
</tr>
</tbody>
</table>

Return Value

| status   | integer | Refer to Appendix A for error codes. |

GetSharedMenuBarEventPanel

```c
int panelHandle = GetSharedMenuBarEventPanel (void);
```

Purpose

This function returns the handle for the panel on which the last menu commit event occurred.

This function is especially useful when one menu bar is shared by multiple panels.
Return Value

| panelHandle | integer | Returns the handle for the panel on which the last commit event occurred. Refer to Appendix A for error codes. |

---

GetSleepPolicy

```c
int sleepPolicy = GetSleepPolicy(void);
```

Purpose

Obtains the current "sleep" policy. See SetSleepPolicy for more information on "sleeping".

Return Value

| sleepPolicy | integer | The current sleep policy. |

Return Codes

| VAL_SLEEP_NONE | 1 | Never be put to sleep. |
| VAL_SLEEP_SOME | 2 | Be put to sleep a small period. |
| VAL_SLEEP_MORE | 3 | Be put to sleep for a longer period. |

See Also

SetSleepPolicy.

---

GetSystemAttribute

```c
int status = GetSystemAttribute(int systemAttribute, void *attributeValue);
```

Purpose

Returns the value of a particular system attribute.
Parameters

| Input   | systemAttribute | integer | ATTR_ALLOW_UNSAFE_TIMER_EVENTS
|         |                 |         | ATTR_REPORT_LOAD_FAILURE
|         |                 |         | ATTR_ALLOW_MISSING_CALLBACKS
| Output  | attributeValue  | void *  | The value of the specified attribute. See Table 3-26 for values associated with this attribute.

Return Value

| status | integer | Refer to Appendix A for error codes.

GetSystemPopupsAttribute

```c
int status = GetSystemPopupsAttribute(int popupAttribute, void *attributeValue);
```

Purpose

Returns the value of a particular system pop-up attribute.

Parameters

| Input   | popupAttribute | integer | ATTR_MOVABLE or
|         |                |         | ATTR_SYSTEM_MENU_VISIBLE
| Output  | attributeValue | void*   | The value of the specified attribute. See Table 3-2 in Chapter 3 for valid values associated with these attributes.

Return Value

| status | integer | Refer to Appendix A for error codes.

GetTextBoxLine

```c
int status = GetTextBoxLine(int panelHandle, int controlID, int lineIndex, char destinationBuffer[]);
```

Purpose

This function returns a string from the specified text box line into Destination Buffer.
### GetTextBoxLineLength

```c
int status = GetTextBoxLineLength(int panelHandle, int controlID, int lineIndex, int *length);
```

**Purpose**

This function returns the number of characters in the selected text box line (including null characters.)

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td></td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>lineIndex</td>
<td>integer</td>
<td></td>
<td>The zero-based index into the text box.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>integer</td>
<td></td>
<td>Returns number of characters in the selected text box line.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

GetTextDisplaySize

```c
int status = GetTextDisplaySize(char text[], char metaFont[], int *height,
                                 int *width);
```

Purpose

Returns the height and width of the specified text having the specified meta font.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>text</th>
<th>string</th>
<th>The text string whose size is to be calculated.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>metaFont</td>
<td>string</td>
<td>The meta font to be used in calculating the size of the text.</td>
</tr>
<tr>
<td>Output</td>
<td>height</td>
<td>integer</td>
<td>The height of the text in pixels.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>The width of the text in pixels.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

GetTraceAttribute

```c
int status = GetTraceAttribute(int panelHandle, int controlID, int traceNumber,
                                int traceAttribute, int *attributeValue);
```

Purpose

Obtains one of the following strip chart trace attributes: color, trace style, point style, line style, visible.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>traceNumber</td>
<td>integer</td>
<td>The strip chart trace number may be from 1 to the number of defined strip chart traces. The number of defined strip chart traces is established through editing the strip chart in the User Interface Editor or calling SetCtrlAttribute.</td>
</tr>
<tr>
<td></td>
<td>traceAttribute</td>
<td>integer</td>
<td>Selects a particular strip chart trace attribute. See Table 3-20 in Chapter 3 for a complete listing of trace attributes.</td>
</tr>
</tbody>
</table>

Output | attributeValue | integer | The current value of the trace attribute. |

Return Value

| Status | integer | Refer to Appendix A for error codes. |

GetUILLErrorString

char *message = GetUILLErrorString(int errorNum)

Purpose

Converts the error number returned by a User Interface Library function into a meaningful error message.

Parameters

Input | errorNum | integer | Status returned by User Interface function. |

Return Value

| message | string | Explanation of error. |
GetUserEvent

```c
int status = GetUserEvent (int waitMode, int *panelorMenuBarHandle,
                          int *controlorMenuItemID);
```

**Purpose**

Obtains the next commit event or programmer-defined event from the GetUserEvent queue. A commit event occurs when the user changes the state of a hot or validate control or selects a menu item. Programmer-defined events are placed in the GetUserEvent queue by QueueUserEvent. The function description for QueueUserEvent describes valid programmer-defined events.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>waitMode</th>
<th>integer</th>
<th>If 1, GetUserEvent does not return until a commit event occurs. If 0, GetUserEvent returns immediately, whether or not a commit event has occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>panelorMenuBarHandle</td>
<td>integer</td>
<td>Returns the handle of the panel or menu bar on which the commit event occurred. If waitMode is zero (&quot;no wait&quot;) and no event has occurred, a -1 is returned.</td>
</tr>
<tr>
<td></td>
<td>controlorMenuItemID</td>
<td>integer</td>
<td>Returns the ID of the control or menu item on which the commit event occurred. If waitMode is zero (&quot;no wait&quot;) and no event has occurred, a -1 is returned.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

**Return Codes**

Contains the event retrieved from the GetUserEvent queue. The possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No event.</td>
</tr>
<tr>
<td>1</td>
<td>Commit event occurred.</td>
</tr>
<tr>
<td>1000−10000</td>
<td>Event queued by QueueUserEvent.</td>
</tr>
</tbody>
</table>
GetValueFromIndex

```c
int status = GetValueFromIndex(int panelHandle, int controlID, int itemIndex,
    void *itemValue);
```

Purpose

This function returns the value for the specified list index.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list.</td>
</tr>
</tbody>
</table>

| Output      | itemValue     | void *  | Returns the value of the specified list item. |

Return Value

| status      | integer       | Refer to Appendix A for error codes. |

GetValueLengthFromIndex

```c
int status = GetValueLengthFromIndex(int panelHandle, int controlID,
    int itemIndex, int *length);
```

Purpose

This function returns the length of the value for the specified list item.

This function is valid only for lists with string data type.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list.</td>
</tr>
<tr>
<td>Output</td>
<td>length</td>
<td>integer</td>
<td>Returns the length of the value at the Item Index.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

GetWaitCursorState

int cursorState = GetWaitCursorState(void);

Purpose

Returns the state of the cursor, indicating whether the wait cursor is active or inactive.

Return Value

| status | integer | Refer to Appendix A for error codes. |

Return Codes

Returns the state of the wait cursor.

| 0 | wait cursor inactive. |
| 1 | wait cursor active. |
HidePanel

```c
int status = HidePanel(int panelHandle);
```

**Purpose**

Clears a panel from the screen but leaves it in memory.

When a panel is hidden, it can still be updated programmatically. For example, you can add plots to a graph control on a hidden panel. When the panel displays again, it will show the new plots.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

---

InsertAxisItem

```c
int status = InsertAxisItem(int panelHandle, int controlID, int axis,
                             int itemIndex, char itemLabel[], double itemValue);
```

**Purpose**

This function inserts a string/value pair into the list of label strings associated with a graph or strip chart axis. These strings appear instead of the numerical labels. They appear at the location of their associated values on the graph or strip chart.

To see string labels on an X axis, you must set the ATTR_XUSE_LABEL_STRINGS attribute to TRUE. To see string labels on a Y axis, you must set the ATTR_YUSE_LABEL_STRINGS attribute to TRUE.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>PanelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>axis</td>
<td>integer</td>
<td>Specifies the axis for which to insert the string/value pair. Valid values: VAL_XAXIS, VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS (graphs only)</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list at which the item is to be stored. Pass -1 to store the item at the end of the list. See discussion below.</td>
</tr>
<tr>
<td></td>
<td>itemLabel</td>
<td>string</td>
<td>The label string being inserted. The label appears on the axis at the location of the associated value. A maximum of 31 characters are shown.</td>
</tr>
<tr>
<td></td>
<td>itemValue</td>
<td>double-precision</td>
<td>The value to be associated with the label string being inserted. The label string appears on the axis at the location of the value.</td>
</tr>
</tbody>
</table>

Return Value

| status  | integer | Refer to Appendix A for error codes. |

Parameter Discussion

itemIndex does not determine the order in which the labels appear on the axis. It merely represents the order in which LabWindows/CVI stores the string/value pairs. You can use the index as a handle for replacing or deleting label/value pairs.

If you pass -1 for itemIndex, the string/value pair is stored at the end of the list.

See Also

ReplaceAxisItem, DeleteAxisItem, ClearAxisItems, GetNumAxisItems
InsertListItem

```c
int status = InsertListItem(int panelHandle, int controlID, int itemIndex,
                           char itemLabel[],...);
```

**Purpose**

This function inserts a label/value pair into a list or ring control at the specified zero-based index.

The indices of existing label/value pairs at and beyond the specified index are increased by 1.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list where the item will be placed. Pass -1 to insert the item at the end of the list.</td>
</tr>
<tr>
<td></td>
<td>itemLabel</td>
<td>string</td>
<td>The label to be associated with the value being inserted.</td>
</tr>
<tr>
<td></td>
<td>itemValue</td>
<td></td>
<td>The value to be associated with the label being inserted. The data type must be the same as the data type of the control.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

**Parameter Discussion**

For picture rings, the “label” is actually an image, and you pass the pathname of the image as the itemLabel parameter. The image pathname may be a complete pathname or a simple filename. If a simple filename, the image is located in the project. If not located in the project, it is located in the directory of the project. If you pass NULL or the empty string, a placeholder for the image is created, which can be filled using ReplaceListItem or SetImageBits.

You can create columns in a list box control by embedding escape codes in the itemLabel string. Use \033 to indicate an escape code, followed by p (for pixel), followed by a justification code
of l (left), c (center), or r (right). For example, the following code sets a left-justified column at the 100- and 200-pixel positions.

```
InsertListItem (handle, cID, 0, "Chevrolet\033p100lCorvette\033p200lRed", 0);
InsertListItem (handle, cID, 1, "Ford\033p100lProbe\033p200lBlack", 0);
```

The preceding code segment creates the following tab format.

<table>
<thead>
<tr>
<th>Chevrolet</th>
<th>Corvette</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ford</td>
<td>Probe</td>
<td>Black</td>
</tr>
</tbody>
</table>

In the following example, the code segment sets a centered column at the 32-, 130-, and 230-pixel positions.

```
InsertListItem (handle, cID, 0, "\033p32cChevrolet\033p130cCorvette\033p230cRed", 0);
InsertListItem (handle, cID, 1, "\033p32cFord\033p130cProbe\033p230cBlack", 0);
```

The preceding code segment creates the following tab format.

<table>
<thead>
<tr>
<th>Chevrolet</th>
<th>Corvette</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ford</td>
<td>Probe</td>
<td>Black</td>
</tr>
</tbody>
</table>

The code that inserts a vertical line is \033vline. The vertical line is added at the current position in the label.

You can also use escape codes to change the foreground and background colors of characters in a list box. The escape codes only affect the label that they are embedded into. Setting the color changes it until the color is changed again or the end of the line is reached.

To change the foreground color, the code is \033fgXXXXXX where XXXXXX is a 6 digit RGB value specified in hex.

To change the background color, the code is \033bgXXXXXX where XXXXXX is a 6 digit RGB value specified in hex.

To change either color back to the default color for the text, insert default instead of the 6 digit RGB value.

You can insert a separator bar into a ring control (Ring, Menu Ring, Recessed Menu Ring, or Pop-up Menu Ring) by embedding the escape code \033m- in the itemLabel string.
InsertSeparator

```c
int status = InsertSeparator(int menuBarHandle, int menuID,
                               int beforeMenuItemID);
```

**Purpose**

This function inserts a new separator bar in the menu.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuBarHandle</td>
<td>integer</td>
<td>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by <code>LoadMenuBar</code> or <code>NewMenuBar</code>.</td>
</tr>
<tr>
<td>menuID</td>
<td>integer</td>
<td>The ID for a particular menu within a menu bar. The Menu ID should be a constant name (located in the .uir header file) generated by the User Interface Editor or a value returned by the <code>NewMenu</code> function.</td>
</tr>
<tr>
<td>beforeMenuItemID</td>
<td>integer</td>
<td>The separator bar is inserted before (above) this menu item. To place the separator at the end (bottom) of the menu item list, enter -1 as the Before Menu Item ID.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

InsertTextBoxLine

```c
int status = InsertTextBoxLine(int panelHandle, int controlID, int lineIndex, char text[]);
```

**Purpose**

This function inserts a string at the specified text box line.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</td>
</tr>
</tbody>
</table>
### controlID
integer
The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.

### lineIndex
integer
The zero-based index of a line in the specified text box. Pass -1 to insert the item at the end of the text box.

### text
string
The text to be inserted at the specified Line Index.

#### Return Value

| status | integer | Refer to Appendix A for error codes. |

#### Parameter Discussion

A text box in the User Interface Library can have only a limited number of lines. The number of lines multiplied by the pixel height of the font must be less than 32767, or else the text box will not scroll. In a text box that uses NIDialogMetaFont or NIEditorMetaFont, you can have a maximum of about 2500 lines.

---

**InstallCtrlCallback**

```c
int status = InstallCtrlCallback (int panelHandle, int controlID, CtrlCallbackPtr eventFunction, void *callbackData);
```

#### Purpose

This function installs a control callback. The `InstallCtrlCallback` function call takes a panel handle and control ID, the name of the event function that processes events for that control, and callback data of any type.

When an event is generated by the specified control (after the callback is installed), the event function is called.

This event function (type `CtrlCallbackPtr`) takes the form:

```c
int CVICALLBACK EventFunctionName (int panelHandle, int controlID, int event, void *callbackData, int eventData1, int eventData2);
```

The event function is passed the panel handle and control ID of the control generating the event. Callback data defined by the user is also passed to the event function as well as the type of the event (such as a left mouse click), and any additional event data (such as the mouse position at
the time of a left mouse click). The possible control events and associated event data are shown in Table 3-1 of Chapter 3, *Programming with the User Interface Library*. Also refer to the online help for the Event Function control on the `InstallCtrlCallback` function panel.

A control callback can also be installed from the Edit Control panel in the User Interface Editor.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>eventFunction</td>
<td>CtrlCallbackPtr</td>
<td>The name of the user function that processes the control callback.</td>
</tr>
<tr>
<td></td>
<td>callbackData</td>
<td>void *</td>
<td>A pointer to user defined data that is passed to the callback function.</td>
</tr>
</tbody>
</table>

**Return Value**

| status     | integer       | Refer to Appendix A for error codes. |

**InstallMainCallback**

```c
int status = InstallMainCallback (MainCallbackPtr eventFunction,  
void *callbackData, int getIdleEvents);
```

**Purpose**

This function installs a main callback that is called for all panel, control, and menu events.

The `InstallMainCallback` function takes the name of an event function a pointer to callback data of any type, and a Boolean indicating whether or not to get idle events. (For a further discussion of the main callback, see the class help for Callback Functions.)

The `getIdleEvents` feature allows program execution to proceed when it would normally be suspended. For example, the program would normally be suspended in `GetUserEvent` or `RunUserInterface` when the user holds the mouse button down on a control or pull-down menu. Idle events occur continuously, even under these circumstances. Since the main callback
can receive idle events, the program can continue processing by responding to idle events. Call `SetIdleEventRate` to set the rate of idle events.

**Note:** `EVENT_VAL_CHANGED` and `EVENT_IDLE` are the only events that are generated while a user holds the mouse button down on a control or pull-down menu.

The operating system blocks all events (including idle events) when a top-level panel is moved or sized.

When any event is generated (after the callback is installed), the event function is called.

This event function (type `MainCallbackPtr`) takes the form:

```c
int CVICALLBACK EventFunctionName (int panelOrMenuBarHandle, int controlOrMenuItemID, int event, void *callbackData, int eventData1, int eventData2);
```

The event function is passed information on the source of the callback, either the panel handle and control ID, or the menu bar handle and menu item ID. Callback data defined by the user is also passed to the event function. Finally, the event function is also passed the type of the event (such as a left mouse click), and any additional event data (such as the mouse position at the time of a left mouse click). The possible events and associated event data that can be processed in the main callback are listed in Table 3-1 of Chapter 3, *Programming with the User Interface Library*. Also refer to the online help for the Event Function control on the `InstallMainCallback` function panel.

### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>eventFunction</th>
<th>MainCallbackPtr</th>
<th>The name of the user function that processes the main callback.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>callbackData</td>
<td>void *</td>
<td>A pointer to user-defined data passed to the event function.</td>
</tr>
<tr>
<td></td>
<td>getIdleEvents</td>
<td>integer</td>
<td>1 = respond to idle events.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = ignore idle events.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>
InstallMenuCallback

```c
int status = InstallMenuCallback (int menuBarHandle, int menuorMenuID, MenuCallbackPtr eventFunction, void *callbackData);
```

**Purpose**

This function installs a menu callback. It takes a menu bar handle and menu/menu item ID, the name of the event function that processes commit events for the menu/menu item, and callback data of any type.

When an event is generated on the specified menu bar (after the callback is installed), the event function is called.

This event function (type MenuCallbackPtr) takes the form:

```c
void CVICALLBACK EventFunctionName (int menuBarHandle, int menuID, void *callbackPtr, int panelHandle);
```

When a commit event is generated by a menu selection, the event function is passed the menu bar handle, menu item ID, and panel handle of the menu bar generating the event. Callback data defined by the user is also passed to the event function.

A menu callback can also be installed via the NewMenuItem function, or from the Edit Menu Bar dialog box in the User Interface Editor.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>menuorMenuID</td>
<td>integer</td>
<td>The menu or menu item ID assigned in the User Interface Editor or returned by the NewMenu or NewMenuItem.</td>
</tr>
<tr>
<td></td>
<td>eventFunction</td>
<td>MenuCallbackPtr</td>
<td>The name of the user function that processes the menu callback.</td>
</tr>
<tr>
<td></td>
<td>callbackData</td>
<td>void *</td>
<td>A pointer to user-defined data that is passed to the event function.</td>
</tr>
</tbody>
</table>

**Return Value**

| status      | integer | Refer to Appendix A for error codes. |
InstallMenuDimmerCallback

```c
int status = InstallMenuDimmerCallback (int menuBarHandle,
                                               MenuDimmerCallbackPtr dimmerFunction);
```

**Purpose**

This function installs a menu dimmer callback. It takes the handle of a menu bar and the name of the callback function.

A menu dimmer callback is called just before a pull-down menu appears when a menu bar is clicked or a menu short-cut key is pressed. The menu bar handle and the handle of the panel on which the menu bar resides is passed to the specified dimmer function for processing. This callback function is useful for checking a program’s state and then dimming invalid menu items before the menu is displayed.

This callback function (type MenuDimmerCallbackPtr) takes the form:

```c
void CVICALLBACK DimmerFunctionName (int menuBarHandle, int panelHandle);
```

The dimmer function is passed the menu bar handle and panel handle of the menu bar that has been accessed.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dimmerFunction</td>
<td>MenuDimmerCallbackPtr</td>
<td>The name of the user function that processes the menu dimmer callback.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

InstallPanelCallback

```c
int status = InstallPanelCallback (int panelHandle, PanelCallbackPtr eventFunction, void *callbackData);
```
Purpose

This function installs a panel callback. It takes a panel handle, the name of the event function that processes events for that panel, and callback data of any type.

When an event is generated on the specified panel (after the callback is installed), the event function is called.

This event function (type `PanelCallbackPtr`) takes the form:

```c
int CVICALLBACK EventFunctionName (int panelHandle, int event,
                                     void *callbackData, int eventData1,
                                     int eventData2);
```

The event function is passed the panel handle of the panel generating the event, callback data defined by the user, the type of the event (such as a left mouse click), and any additional event data (such as the mouse position at the time of a left mouse click).

See Table 3-1 in Chapter 3, Programming with the User Interface Library, for the events and event data that can be processed by a panel callback. Also refer to the online help for the Event Function control on the `InstallPanelCallback` function panel.

A panel callback can also be installed from the Edit Panel window in the User Interface Editor.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventFunction</td>
<td>PanelCallbackPtr</td>
<td>void *</td>
<td>The name of the user function that processes the panel callback. A pointer to user defined data that is passed to the event function.</td>
</tr>
<tr>
<td>callbackData</td>
<td>void *</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

InstallPopup

```c
int status = InstallPopup (int panelHandle);
```
Purpose

Displays and activates a panel as a dialog box.

The user cannot operate any other panels while a pop-up panel is active.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

IsListItemChecked

```c
int status = IsListItemChecked (int panelHandle, int controlID, int itemIndex, int *checked);
```

Purpose

This function returns a Boolean value indicating whether or not a list item is checked.

This function only applies to list boxes with the check mode attribute set.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
<td></td>
</tr>
<tr>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list control.</td>
<td></td>
</tr>
</tbody>
</table>
Output | checked | integer | Returns a Boolean value indicating whether a list item is checked.  
|        |         |        | 0 = not checked  
|        |         |        | 1 = checked.

Return Value

| status | integer | Refer to Appendix A for error codes. |

LoadMenuBar

int menuBarHandle = LoadMenuBar(int destinationPanelHandle, char filename[], int menuBarResourceID);

Purpose

Loads a menu bar into memory from a User Interface Resource (.uir) file that was created in the User Interface Editor. The menu bar will reside on the panel specified by the destination panel handle.

The function returns a menu bar handle which is used in subsequent function calls to specify the menu bar.

Note: The ATTR_REPORT_LOAD_FAILURE and ATTR_ALLOW_MISSING_CALLBACKS system attribute affect the behavior of this function when an error is encountered. See the SetSystemAttribute function.

Parameters

| Input       | destinationPanelHandle | integer | The handle for the panel on which the menu bar is to reside. |
|            | filename               | string  | The name of the User Interface Resource file which contains the menu bar. |
|            | menuBarResourceID      | integer | The defined constant which was assigned to the menu bar in the User Interface Editor. |

Return Value

| menuBarHandle | integer | The value you can use in subsequent function calls to specify this menu bar. Refer to Appendix A for error codes. |

Parameter Discussion
**destinationPanelHandle** will have been returned by the **LoadPanel**, **NewPanel**, or **DuplicatePanel** function. If the destination panel is not currently in memory, pass a zero as the destination panel handle and this menu bar can later be assigned to a panel using **SetPanelMenuBar**.

You can use a complete pathname or a simple filename with the **filename** parameter. If the name is a simple filename (in other words, contains no directory path) and is listed in the project, then the file is loaded using the pathname from the project. Otherwise, the file is loaded from the directory containing the project.

The **menuBarResourceID** can be found in the .uir header file and is used only once to load the menu bar into memory. Subsequent function calls will refer to the menu bar with the menu bar handle returned by this function.

**See also**

LoadMenuBarEx, SetSystemAttribute

---

**LoadMenuBarEx**

```c
int menuBarHandle = LoadMenuBarEx(int destinationPanelHandle, filename[], int menuBarResourceID, void *callingModuleHandle);
```

**Purpose**

LoadMenuBarEx loads a menu bar into memory from a User Interface Resource (.uir) that was file created in the User Interface Editor. LoadMenuBarEx is similar to LoadMenuBar, except that, when you use LoadMenuBarEx on Windows 95 and NT, the callback functions you reference in your .uir file can be defined in the DLL that contains the call to LoadMenuBarEx. On platforms other than Windows 95 and NT, LoadMenuBarEx works exactly like LoadMenuBar.

**Note:** The **ATTR_REPORT_LOAD_FAILURE** and **ATTR_ALLOW_MISSING_CALLBACKS** system attribute affect the behavior of this function when an error is encountered. See the **SetSystemAttribute** function.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>destinationPanelHandle</strong></th>
<th>Integer</th>
<th>The handle of the panel on which the menu bar is to reside.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>filename</strong></td>
<td></td>
<td>String</td>
<td>The name of the User Interface Resource File that contains the menu bar.</td>
</tr>
</tbody>
</table>
**panelResourceId**
integer
The defined constant assigned to the menu bar in the User Interface Editor.

**callingModuleHandle**
void pointer
Usually, the module handle of the calling DLL. You can use _CVIUInst. Zero indicates the project or executable.

---

**Return Value**

| **menuItemHandle** | integer | The value you can use in subsequent function calls to specify this menu bar. Negative values indicate that an error occurred. Refer to Appendix A for error codes. |

---

**Using this Function**

Refer to the function help for **LoadMenuBar** for detailed information on that function. When you call **LoadMenuBar**, the User Interface Library attempts to find the callback functions referenced in the .uir file. It searches the symbols defined in the project or in object, library, or DLL import library modules that have been loaded using **LoadExternalModule**. It does not search symbols that are defined in a DLL but not exported in the DLL import library.

You may want a DLL to load a menu bar and link to callback functions defined in (but not exported from) the DLL. You can do this by calling **LoadMenuBarEx**. You must specify the module handle of the DLL in the **callingModuleHandle** parameter. You can do this by using the pre-defined variable _CVIUInst. (If you pass zero for the **callingModuleHandle**, the function behaves identically to **LoadMenuBar**.)

**LoadMenuBarEx** first searches the DLL symbols to find the callback functions referenced in the .uir. If there are any callback functions that it cannot find, it then searches for them in the same manner as **LoadMenuBar**.

**LoadMenuBarEx** expects the DLL to contain a table of the callback functions referenced by the .uir files loaded by the DLL. If you create the DLL in LabWindows/CVI, the table is included automatically. If you create the DLL using an external compiler, you must arrange for this table to be included in the DLL. You can do this by using the **External Compiler Support** command in the **Build** menu of the Project window. You must have a LabWindows/CVI project that lists all of the .uir files loaded by the DLL. In the External Compiler Support dialog box, specify the name of an object file to contain the table of callback function names. Then click on the **Create** button to create the object file. You must include the object file in the external compiler project you use to create the DLL.

The External Compiler Support information is contained in the LabWindows/CVI project file. If that project file is loaded and you modify and save any of the .uir files, LabWindows/CVI automatically regenerates the object file.
LoadPanel

int panelHandle = LoadPanel(int parentPanelHandle, char filename[], int panelResourceID);

Purpose

Loads a panel into memory from a User Interface Resource (.uir) file created in the User Interface Editor.

The panel will be a child panel of the parent panel specified by parentPanelHandle. To make the panel a top-level panel, enter 0 for the parentPanelHandle. The loaded panel can then be displayed with the DisplayPanel function.

The function returns a panel handle which is used in subsequent function calls to specify the panel.

Note: The ATTR_REPORT_LOAD_FAILURE and ATTR_ALLOW_MISSING_CALLBACKS system attribute affect the behavior of this function when an error is encountered. See the SetSystemAttribute function.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>parentPanelHandle</th>
<th>integer</th>
<th>The handle of the panel into which the panel is loaded as a child panel. Pass 0 to load the panel as a top-level window.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>filename</td>
<td>string</td>
<td>The name of the User Interface Resource file which contains the panel.</td>
</tr>
<tr>
<td></td>
<td>panelResourceID</td>
<td>integer</td>
<td>The defined constant which was assigned to the panel in the User Interface Editor.</td>
</tr>
</tbody>
</table>

Return Value

|                  | panelHandle | integer | The value you can use in subsequent function calls to specify this panel. Negative values indicate that an error occurred. Refer to Appendix A for error codes. |

Parameter Discussion
To make the panel a top-level panel, enter 0 for `parentPanelHandle`. If the panel is being loaded from a LabWindows for DOS `.uir` file, use the Return Value from the function `DOSCompatWindow` as the `parentPanelHandle`.

You can use a complete pathname or a simple filename for `filename`. If the name is a simple filename (in other words, contains no directory path) and is listed in the project, then the file is loaded using the pathname from the project. Otherwise, the file is loaded from the directory containing the project.

The `panelResourceID` can be found in the `.uir` header file and is used only once to load the panel into memory. Subsequent function calls refer to this panel with the `panelHandle` returned by this function.

**See also**

`LoadPanelEx`, `SetSystemAttribute`

---

### LoadPanelEx

```c
int panelHandle = LoadPanelEx(int parentPanelHandle, char filename[], int panelResourceID, void *callingModuleHandle);
```

**Purpose**

`LoadPanelEx` loads a panel into memory from a User Interface Resource (.uir) file created in the User Interface Editor. `LoadPanelEx` is similar to `LoadPanel`, except that, when you use `LoadPanelEx` your program on Windows 95 and NT, the callback functions you reference in your `.uir` file can be defined in the DLL that contains the call to `LoadPanelEx`. On platforms other than Windows 95 and NT, `LoadPanelEx` works exactly like `LoadPanel`.

**Note:** The `ATTR_REPORT_LOAD_FAILURE` and `ATTR_ALLOW_MISSING_CALLBACKS` system attribute affect the behavior of this function when an error is encountered. See the `SetSystemAttribute function`.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>parentPanelHandle</code></th>
<th>integer</th>
<th>The handle of the panel into which the panel is loaded as a child panel. Pass 0 to load the panel as a top-level window.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>filename</code></td>
<td>string</td>
<td>The name of the User Interface Resource File that contains the panel.</td>
</tr>
<tr>
<td></td>
<td><code>panelResourceID</code></td>
<td>integer</td>
<td>The defined constant assigned to the panel in the User Interface Editor.</td>
</tr>
</tbody>
</table>
callingModuleHandle | void pointer | Usually, the module handle of the calling DLL. You can use __CVIUserHInst. Zero indicates the project or executable.

Return Value

panelHandle | integer | The value you can use in subsequent function calls to specify this panel. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

Using this Function

Refer to the function help for LoadPanel for detailed information on that function.

When you call LoadPanel, the User Interface Library attempts to find the callback functions referenced in the .uir file. It searches the symbols defined in the project or in object, library, or DLL import library modules that have been loaded using LoadExternalModule. It does not search symbols that are defined in a DLL but not exported in the DLL import library.

You may want a DLL to load a panel and link to callback functions defined in (but not exported from) the DLL. You can do this by calling LoadPanelEx. You must specify the module handle of the DLL in the callingModuleHandle parameter. You can do this by using the pre-defined variable __CVIUserHInst. (If you pass zero for the callingModuleHandle, the function behaves identically to LoadPanel.)

LoadPanelEx first searches the DLL symbols to find the callback functions referenced in the .uir. If there are any callback functions that it cannot find, it then searches for them in the same manner as LoadPanel.

LoadPanelEx expects the DLL to contain a table of the callback functions referenced by the .uir files loaded by the DLL. If you create the DLL in LabWindows/CVI, the table is included automatically. If you create the DLL using an external compiler, you must arrange for this table to be included in the DLL. You can do this by using the External Compiler Support command in the Build menu of the Project window. You must have a LabWindows/CVI project that lists all of the .uir files loaded by the DLL. In the External Compiler Support dialog box, specify the name of an object file to contain the table of callback function names. Then click on the Create button to create the object file. You must include the object file in the external compiler project you use to create the DLL.

The External Compiler Support information is contained in the LabWindows/CVI project file. If that project file is loaded and you modify and save any of the .uir files, LabWindows/CVI automatically regenerates the object file.
See also

LoadPanel, LoadMenuBarEx, SetSystemAttribute

---

**MakeColor**

```c
int rgb = MakeColor(int red, int green, int blue);
```

**Purpose**

Generates a color (RGB) value from the individual constituent red, green, and blue intensity levels.

An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. The first sixteen colors listed are the sixteen standard colors.

**Predefined RGB Values:**

```
0xFF0000L = VAL_RED
0x00FF00L = VAL_GREEN
0x0000FFL = VAL_BLUE
0x00FFFFL = VAL_CYAN
0xFFFF00L = VAL_MAGENTA
0xFF00FFL = VAL_YELLOW
0x800000L = VAL_DK_RED
0x000080L = VAL_DK_BLUE
0x008000L = VAL_DK_GREEN
0x008080L = VAL_DK_CYAN
0x808080L = VAL_DK_MAGENTA
0x808000L = VAL_DK_YELLOW
0xCCCCCL = VAL_LT_GRAY
0x808080L = VAL_DK_GRAY
0x000000L = VAL_BLACK
0xFFFFFL = VAL_WHITE
0xA0A0A0L = VAL_GRAY
0xE5E5E5L = VAL_OFFWHITE
VAL_PANEL_GRAY = VAL_LT_GRAY
```

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>parameter</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>integer</td>
<td></td>
<td>The red level of the new color. Any integer value between 0 and 255 is valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = No red component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>255 = Maximum red component.</td>
</tr>
<tr>
<td>green</td>
<td>integer</td>
<td></td>
<td>The green level of the new color. Any integer value between 0 and 255 is valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = No green component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>255 = Maximum green component.</td>
</tr>
<tr>
<td>blue</td>
<td>integer</td>
<td>The blue level of the new color. Any integer value between 0 and 255 is valid. 0 = No blue component. 255 = Maximum blue component.</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

| rgb   | integer | Return Value indicating the RGB value. Refer to Appendix A for error codes. |

MakePoint

Point  point = MakePoint (int  xCoordinate, int  yCoordinate);

Purpose

Returns a Point structure with the specified values. The Point structure defines the location of a point.

This function is useful when calling canvas control functions that require Point structures as input parameters. You can embed a call to MakePoint in calls to these functions, thereby eliminating the need to declare a Point variable.

Parameters

| Input | xCoordinate | integer | The horizontal location of the point. |
| yCoordinate | integer | The vertical location of the point. |

Return Value

| point | Point | A Point structure containing the specified coordinate values. |

See Also

PointSet, MakeRect
MakeRect

Rect rect = MakeRect(int top, int left, int height, int width);

Purpose

Returns a Rect structure with the specified values. The Rect structure defines the location and size of a rectangle.

This function is useful when calling canvas control functions that require Rect structures as input parameters. You can embed a call to MakeRect in calls to these functions, thereby eliminating the need to declare a Rect variable.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>top</th>
<th>integer</th>
<th>The location of the top edge of the rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>left</td>
<td>integer</td>
<td>The location of the left edge of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>integer</td>
<td>The height of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>The width of the rectangle.</td>
</tr>
</tbody>
</table>

Return Value

| rect | Rect | A Rect structure containing the specified coordinate values. |

See Also

RectSet, MakePoint

MessagePopup

int status = MessagePopup(char title[], char message[]);

Purpose

Displays a message in a dialog box and waits for the user to select the OK button.

Parameters

| Input | title | string | The title to be displayed on the dialog box. |
The message to be displayed in the dialog box. To display a multi-line message, embed newline characters (\n) in the message string.

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
</tr>
</tbody>
</table>
MultiFileSelectPopup

```c
int status = MultiFileSelectPopup(char defaultDirectory[], char defaultFileSpec[],
char fileTypeList[], char title[], int restrictDirectory, int restrictExtension, int allowCancel,
int numberOfSelectedFiles, char **fileList);
```

**Purpose**

Displays a file-selection dialog box and waits for the user to select a file or cancel.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>defaultDirectory</code></td>
<td>string</td>
<td>The initial directory. If &quot;&quot; is entered, then the current working directory will be used.</td>
</tr>
<tr>
<td><code>defaultFileSpec</code></td>
<td>string</td>
<td>A string that specifies which files to display. For example, &quot;*.c&quot; would cause all files with the extension .c to be displayed.</td>
</tr>
<tr>
<td><code>fileTypeList</code></td>
<td>string</td>
<td>A list of file types, separated by semicolons, to be contained in the File Type List of the File Select Pop-up when <code>restrictExtension</code> is FALSE. For example, &quot;<em>.c;</em>.h&quot; allows the user to select &quot;<em>.c&quot; or &quot;</em>.h&quot; from the File Type List. (&quot;<em>.</em>&quot; is always available).</td>
</tr>
<tr>
<td><code>title</code></td>
<td>string</td>
<td>The title of the dialog box.</td>
</tr>
<tr>
<td><code>restrictDirectory</code></td>
<td>integer</td>
<td>If non-zero, the user cannot change directories or drives. If zero, the user can change directories or drives.</td>
</tr>
<tr>
<td><code>restrictExtension</code></td>
<td>integer</td>
<td>If non-zero, the user is limited to files with the default extension. If zero, the user can select files with any extension.</td>
</tr>
<tr>
<td><code>allowCancel</code></td>
<td>integer</td>
<td>If non-zero, the user can cancel out of the File Select Popup. If zero, the user can leave the pop-up only by making a selection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>numberOfSelectedFiles</code></td>
<td>integer</td>
<td>The number of files selected by the user.</td>
</tr>
<tr>
<td><code>fileList</code></td>
<td>array of strings</td>
<td>The buffer in which the user’s selection is returned. The buffer is automatically allocated by MultiFileSelectPopup and is accessible as an array of strings.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

Return Codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VAL_NO_FILE_SELECTED</td>
</tr>
<tr>
<td>1</td>
<td>VAL_EXISTING_FILE_SELECTED</td>
</tr>
</tbody>
</table>

Parameter Discussion

The `defaultFileSpec` is shown in the file name control when the pop-up is first displayed. If you specify an actual file name, such as `test.c`, that name appears in the file name box and also in the file list box. The `defaultFileSpec` cannot contain a directory path.

`fileList` is allocated dynamically, as is each string. When they are no longer needed, your program should free each string, and then free the array using the `free` function.

---

**NewBitmap**

```c
int status = NewBitmap (int bytesPerRow, int pixelDepth, int width, int height,
                        int colorTable[], unsigned char bits[],
                        unsigned char mask[], int *bitmapID);
```

Purpose

Creates a bitmap object. The bitmap ID can then be passed any function that accepts a bitmap, such as `CanvasDrawBitmap` or `ClipboardPutBitmap`.

You can discard the bitmap object by passing its ID to `DiscardBitmap`. 
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>bytePerRow</th>
<th>integer</th>
<th>The number of bytes on each scan line of the image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixelDepth</td>
<td>integer</td>
<td></td>
<td>The number of bits per pixel.</td>
</tr>
<tr>
<td>width</td>
<td>integer</td>
<td></td>
<td>The width of the image, in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>integer</td>
<td></td>
<td>The height of the image, in pixels.</td>
</tr>
<tr>
<td>colorTable</td>
<td>integer array</td>
<td>An array of RGB color values, or NULL if pixelDepth is greater than 8.</td>
<td></td>
</tr>
<tr>
<td>bits</td>
<td>unsigned char array</td>
<td>An array of bits that determine the colors to be displayed on each pixel in the image.</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>unsigned char array</td>
<td>An array containing one bit per pixel in the image. Each bit specifies whether the pixel is actually drawn. May be NULL.</td>
<td></td>
</tr>
</tbody>
</table>

Output | bitmapID | integer | An ID that serves as a handle to the bitmap object. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

Parameter Discussion

Depending on the pixelDepth and width, the number of bits per line in the bits array might not be an even multiple of 8. If not, then the extra bits needed to get to the next byte boundary are considered "padding". If you set bytesPerRow to be a positive number, then the bits for each scan line must start on a byte boundary, and so padding may be required. In fact, you can set bytesPerRow to be larger than the minimum number of bytes actually needed. The extra bytes are also considered padding. If you pass -1, there is no padding at all. The bits for each scan line immediately follow the bits for the previous scan line.

The valid values for pixelDepth are 1, 4, 8, 24, and 32.

If the pixelDepth is 8 or less, the number of entries in the colorTable array must equal 2 raised to the power of the pixelDepth parameter. The bits array contain indices into the colorTable array. If the pixelDepth is greater than 8, the colorTable parameter is not used. Instead the bits array contains actual RGB color values, rather than indices into the colorTable array.

For a pixelDepth of 24, each pixel is represented by a 3-byte RGB value of the form 0xRRGGBB, where RR, GG, and BB represent the red, green and blue intensity of the color. The RR byte should always be at the lowest memory address of the three bytes.

If the pixelDepth is 32, each pixel in the bits array is represented by a 32-bit RGB value of the form 0x00RRGGBB, where RR, GG, and BB represent the red, green and blue intensity of the
color. The 32-bit value is treated as a native 32-bit integer value for the platform. The most significant byte is always ignored. The BB byte should always be in the least significant byte. On a little-endian platform (for example, Intel processors), BB would be at the lowest memory address. On a big-endian platform (for example, Motorola processors), BB would be at the highest address. Note that this differs from the format of the bits array when the pixelDepth is 24.

In the mask array, a bit value of 1 indicates that the pixel is drawn. 0 indicates that the pixel is not drawn. Exception: If an image has a pixelDepth of 1, pixels with a bits value of 1 (foreground pixels) are always drawn and the mask affects only the pixels with a bitmap of 0 (background pixels). Each row of the mask must be padded to the nearest even-byte boundary. For example, if the width of the image is 21 pixels, then there must be 32 bits (in other words, 4 bytes) of data in each row of the mask.

A mask is useful for achieving transparency.

You may pass NULL if you do not need a mask.

See Also

NewCtrl

int controlID = NewCtrl (int panelHandle, int controlStyle, char controlLabel [], int controlTop, int controlLeft);

Purpose

This function creates a new control and returns a control ID used to specify the control in subsequent function calls.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
<td></td>
</tr>
<tr>
<td>controlStyle</td>
<td>integer</td>
<td>See Table 3-10 in Chapter 3 for a complete listing of all control styles.</td>
<td></td>
</tr>
<tr>
<td>controlLabel</td>
<td>string</td>
<td>The label of the new control. Pass &quot;&quot; or 0 for no label.</td>
<td></td>
</tr>
<tr>
<td>controlTop</td>
<td>integer</td>
<td>The vertical coordinate at which the upper left corner of the control (not including labels) is placed.</td>
<td></td>
</tr>
<tr>
<td>controlLeft</td>
<td>integer</td>
<td>The horizontal coordinate at which the upper left corner of the control (not including labels) is placed.</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

| controlID | integer | Returns the ID used to specify this control in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes. |

Parameter Discussion

controlTop and controlLeft coordinates must be integer values from -32768 to 32767. The origin (0,0) is at the upper-left corner of the panel (directly below the title bar) before the panel is scrolled.

NewMenu

```c
int menuID = NewMenu(int menuBarHandle, char menuName[], int beforeMenuID);
```

Purpose

This function adds a new menu to the specified menu bar and returns a Menu ID to be used to specify the menu in subsequent function calls.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuName</td>
<td>string</td>
<td></td>
<td>The new menu name that is added to the menu bar.</td>
</tr>
<tr>
<td>beforeMenuID</td>
<td>integer</td>
<td></td>
<td>The new menu will be inserted before this menu. To place the new menu at the end of the menu bar, enter -1 for the beforeMenuID.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | The ID which will be used to reference this menu in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes. |

NewMenuBar

```c
int menuBarHandle = NewMenuBar (int destinationPanelHandle);
```

Purpose

This function creates a new menu bar to reside on the specified destination panel and returns the new menu bar handle.

The new menu bar handle is used in subsequent function calls to specify this menu bar.

Parameters

| Input           | destinationPanelHandle | integer | The handle for the panel on which the menu bar is to reside. |

Return Value

| menuBarHandle | integer | Used in subsequent function calls to specify the menu bar. Refer to Appendix A for error codes. |

Parameter Discussion

If the destination panel is not currently in the memory, pass a zero as the destinationPanelHandle and this menu bar can later be assigned to a panel using SetPanelMenuBar.
NewMenuItem

```c
int menuItemID = NewMenuItem(int menuBarHandle, int menuID,
char itemName[], int beforeMenuItemID,
int shortCutKey, MenuCallbackPtr eventFunction,
void *callbackData);
```

**Purpose**

This function adds a new menu item to the specified menu and returns the MenuItemID to be used in subsequent calls to specify the menu item.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuBarHandle</td>
<td>integer</td>
<td>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar</td>
</tr>
<tr>
<td>menuID</td>
<td>integer</td>
<td>The ID for a particular menu within a menu bar. The Menu ID should be a constant name (located in the .uir header file) generated by the User Interface Editor or a value returned by the NewMenu function.</td>
</tr>
<tr>
<td>itemName</td>
<td>string</td>
<td>The name of the new menu item.</td>
</tr>
<tr>
<td>beforeMenuItemID</td>
<td>integer</td>
<td>The new menu item will be inserted before (above) this menu item. To place the new menu item at the end (bottom) of the menu item list, enter -1 for the beforeMenuItemID.</td>
</tr>
<tr>
<td>shortCutKey</td>
<td>integer</td>
<td>Specifies the key used to automatically select the menu item.</td>
</tr>
<tr>
<td>eventFunction</td>
<td>MenuCallbackPtr</td>
<td>The name of the user function that process the menu item callback.</td>
</tr>
<tr>
<td>callbackData</td>
<td>void *</td>
<td>If you process the menu item event through a callback function, you can supply a pointer to user-defined data in this control. If you do not need to pass user-defined data to the callback function, or if you are not using a callback function to process the menu item event, supply a value of zero.</td>
</tr>
</tbody>
</table>
Return Value

| menuItemID | integer | Returns the ID which will be used to specify this menu item in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes. |

Parameter Discussion

Valid Shortcut Keys:

Shortcut keys are 4-byte integers consisting of three bit fields, 0x00MMVVAA, where:

MM = the modifier key
VV = the virtual key
AA = the ASCII key

When a shortcut key is constructed, modifier keys are bit-wise OR’ed with a virtual key or an ASCII key. Code for either the ASCII field or the virtual key field will be all zeros.

For example:

VAL_SHIFT_MODIFIER | VAL_F1_VKEY

produces a shortcut key of <Shift-F1>, and

VAL_MENUKEY_MODIFIER | 'D'

produces a shortcut key of <Ctrl-D> on the PC and <meta-D> on the SPARCstation. Virtual keys do not require modifiers but ASCII keys require at least one modifier.

The modifier keys are:

VAL_SHIFT_MODIFIER
VAL_MENUKEY_MODIFIER (the <Ctrl> key)
VAL_SHIFT_AND_MENUKEY

The virtual keys are:

VAL_FWD_DELETE_VKEY (not available on the SPARCstation)
VAL_BACKSPACE_VKEY (<Del> on the SPARCstation)
VAL_ESC_VKEY
VAL_TAB_VKEY
VAL_ENTER_VKEY
VAL_UP_ARROW_VKEY
VAL_DOWN_ARROW_VKEY
VAL_LEFT_ARROW_VKEY
The possible ASCII keys are:

'A' or 'a'
'B' or 'b'
'C' or 'c'
.  .  .
'Z' or 'z'

**eventFunction**

If you want to process the menu item event through a callback function, supply the name of that function as this parameter. This event function (type `MenuCallbackPtr`) should be prototyped as follows:

```c
void CVICALLBACK EventFunctionName(int menuBarHandle, int menuItemID, void *callbackData, int panelHandle);
```

The event function is passed the menu bar handle, `menuItemID`, and panel handle of the menu bar generating the callback. Callback data defined by the user is also passed to the event function.

If you do not want to process the menu item event through a callback function, supply a value of zero (0) as this parameter.
NewPanel

```c
int panelHandle = NewPanel (int parentPanelHandle, char panelTitle[], int panelTop, int panelLeft, int panelHeight, int panelWidth);
```

**Purpose**

This function creates a new child panel inside the specified parent panel and returns the new panel handle.

To make the panel a top-level panel, enter 0 for the parent panel handle.

The new panel handle is used in subsequent function calls to specify this panel.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>parentPanelHandle</th>
<th>integer</th>
<th>The handle of the panel into which the new child panel is loaded. To make the panel a top-level panel, enter 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelTitle</td>
<td>string</td>
<td>The title for the new panel.</td>
</tr>
<tr>
<td></td>
<td>panelTop</td>
<td>integer</td>
<td>The vertical coordinate at which the upper left corner of the panel (directly below the title bar) is placed.</td>
</tr>
<tr>
<td></td>
<td>panelLeft</td>
<td>integer</td>
<td>The horizontal coordinate at which the upper left corner of the panel (directly below the title bar) is placed.</td>
</tr>
<tr>
<td></td>
<td>panelHeight</td>
<td>integer</td>
<td>The vertical size of the new panel. The height of the panel does not include the title bar or panel frame.</td>
</tr>
<tr>
<td></td>
<td>panelWidth</td>
<td>integer</td>
<td>Size of the new panel in window coordinates. The width of the panel does not include the title bar or panel frame.</td>
</tr>
</tbody>
</table>

**Return Value**

| panelHandle | integer | Returns a panel handle to be used in subsequent function calls to reference the panel. Refer to Appendix A for error codes. |

**Parameter Discussion**

The `panelTop` and `panelLeft` coordinates must be integer values from -32768 to 32767, or `VAL_AUTO_CENTER` to center the panel. For a top-level panel, (0,0) is the upper-left corner of
the screen. For a child panel, (0,0) is the upper-left corner of the parent panel (directly below the title bar) before the parent panel is scrolled. **panelHeight** and **panelWidth** must be integer values from 0 to 32767.

---

**NewSubMenu**

```c
int subMenuID = NewSubMenu(int menuBarHandle, int menuItemID);
```

**Purpose**

This function creates a submenu for a specified menu item and returns a **subMenuID** to specify the submenu in subsequent function calls.

A submenu appears as a triangle on the right side of the menu item.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>menuItemID</td>
<td>integer</td>
</tr>
</tbody>
</table>

The specifier for a particular menu bar that is currently in memory. This handle will have been returned by `LoadMenuBar` or `NewMenuBar`.

This is the menu item on which the submenu is to be attached. The Menu Item ID will have been a constant name supplied in the User Interface Editor or a value returned by the `NewMenuItem` function.

**Return Value**

<table>
<thead>
<tr>
<th>subMenuID</th>
<th>integer</th>
</tr>
</thead>
</table>

The ID which will be used to specify this submenu in subsequent function calls. Negative values indicate that an error occurred. Refer to Appendix A for error codes.

---

**PlotArc**

```c
int plotHandle = PlotArc(int panelHandle, int controlID, double x1, double y1, double x2, double y2, int beginAngle, int arcAngle, int color, int fillColor);
```

**Purpose**

Plots an arc into a graph control.
The arc is defined by specifying two opposing corners of a rectangle that enclose the arc, along with a beginning angle (in tenths of degrees) and an arc angle (in tenths of degrees).

For example, if \( X1=0.0, Y1=0.0, X2=200.0, Y2=200.0, \text{Beg Angle}=0, \) and \( \text{Arc Angle}=3600, \) then a circle centered on coordinates \((100.0,100.0)\) would be plotted on the graph.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>x1</td>
<td>double-precision</td>
<td></td>
<td>The horizontal coordinate of one corner of the rectangle that encloses the arc.</td>
</tr>
<tr>
<td>y1</td>
<td>double-precision</td>
<td></td>
<td>The vertical coordinate of one corner of the rectangle that encloses the arc.</td>
</tr>
<tr>
<td>x2</td>
<td>double-precision</td>
<td></td>
<td>The horizontal coordinate of the opposing corner of the rectangle that encloses the arc.</td>
</tr>
<tr>
<td>y2</td>
<td>double-precision</td>
<td></td>
<td>The vertical coordinate of the opposing corner of the rectangle that encloses the arc.</td>
</tr>
<tr>
<td>beginAngle</td>
<td>integer</td>
<td></td>
<td>The starting angle of the arc, in tenths of degrees. Positive angles proceed counter-clockwise and negative angles proceed clockwise, from 0 to 3600.</td>
</tr>
<tr>
<td>arcAngle</td>
<td>integer</td>
<td></td>
<td>The sweep angle of the arc, in tenths of degrees. Positive angles proceed counter-clockwise and negative angles proceed clockwise, from 0 to 3600.</td>
</tr>
<tr>
<td>color</td>
<td>integer</td>
<td></td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
<tr>
<td>fillColor</td>
<td>integer</td>
<td></td>
<td>Specifies the fill color of the figure.</td>
</tr>
</tbody>
</table>
Return Value

| plotHandle       | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute |

PlotBitmap

```c
int plotHandle = PlotBitmap (int panelHandle, int controlID, double x, double y,
                                double width, double height, char fileName[]);
```

Purpose

Plots a bitmapped image into a graph control.

The origin of the image is expressed in terms of the X and Y coordinates of the graph. It pinpoints the lower left corner of the image.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>double-precision</td>
<td>The x-coordinate of the lower left corner of the bitmapped image.</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>double-precision</td>
<td>The y-coordinate of the lower left corner of the bitmapped image.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>double-precision</td>
<td>The width, in graph units, of the area in which to fit the bitmapped image. If zero, then the width is the width of the image. If nonzero, the image is stretched or shrunk to fit.</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>double-precision</td>
<td>The height, in graph units, of the area in which to fit the bitmapped image. If zero, then the height is the height of the image. If nonzero, the image is stretched or shrunk to fit.</td>
</tr>
<tr>
<td></td>
<td>fileName</td>
<td>string</td>
<td>The name of the file which contains the image.</td>
</tr>
</tbody>
</table>
Return Value

| plotHandle | integer | The handle for the plot. This handle can be passed to SetImageBits, GetImageBits, SetPlotAttribute, GetPlotAttribute, or DeleteGraphPlot. Refer to Appendix A for error codes. |

Parameter Discussion

fileName can be a complete pathname or a simple filename. If it is a simple filename (in other words, contains no directory path):

- If the file is listed in the project, then the file is loaded using the pathname from the project.
- Otherwise, the file is loaded from the directory containing the project.

The valid image types appear in the following list.

- PCX: Windows and UNIX
- BMP, DIB, RLE, ICO: Windows only
- WMF: Windows 95 and NT only
- XWD: UNIX only

fileName may be NULL or the empty string. This is useful when you want to define the image programmatically by calling SetCtrlBitmap using the PlotHandle returned by this function. The plot is not visible until a valid image is specified.

See Also

SetPlotAttribute, GetPlotAttribute

PlotIntensity

int plotHandle = PlotIntensity (int panelHandle, int controlID, void *zArray, int numberOfXPoints, int numberOfYPoints, int zDataType, ColorMapEntry colorMapArray[], int hiColor, int numberOfColors, int interpColors, int interpPixels);

Purpose

This function draws a solid rectangular plot in a graph control. If you want to apply scaling factors and offsets to the data values, see the PlotScaledIntensity function.

The plot consists of pixels whose colors correspond to the magnitude of data values in a two-dimensional array and whose coordinates correspond to the locations of the data values in the array. For instance, the pixel associated with zArray[2][3] is located at \( \{x=3, y=2\} \).
The size of the lower left corner of the plot area is at \( \{0, 0\} \).

The upper right corner of the plot area is at \( \{X-1, Y-1\} \), where,

\[
X = \text{Number of X Points} \\
Y = \text{Number of Y Points}
\]

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>panelHandle</strong></td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</td>
</tr>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td><strong>zArray</strong></td>
<td>numeric array</td>
<td>An array that contains the data values that are to be converted to colors.</td>
</tr>
<tr>
<td></td>
<td><strong>numberOfXPoints</strong></td>
<td>integer</td>
<td>The number of points to be displayed along the x-axis in each row.</td>
</tr>
<tr>
<td></td>
<td><strong>numberOfYPoints</strong></td>
<td>integer</td>
<td>The number of points to be displayed along the y-axis in each column.</td>
</tr>
<tr>
<td></td>
<td><strong>zDataType</strong></td>
<td>integer</td>
<td>Specifies the data type of the elements in <code>zArray</code>, as well as the data type of the Color Map values.</td>
</tr>
<tr>
<td></td>
<td><strong>colorMapArray</strong></td>
<td>ColorMapEntry array</td>
<td>An array of <code>ColorMapEntry</code> structures which specifies how the data values in <code>zArray</code> are translated into colors. The maximum number of entries is 255.</td>
</tr>
<tr>
<td></td>
<td><strong>hiColor</strong></td>
<td>integer</td>
<td>The RGB value to which all <code>zArray</code> values that are higher than the highest data value in <code>colorMapArray</code> are translated.</td>
</tr>
<tr>
<td></td>
<td><strong>numberOfColors</strong></td>
<td>integer</td>
<td>The number of entries in <code>colorMapArray</code>. Must be less than or equal to 255.</td>
</tr>
<tr>
<td></td>
<td><strong>interpColors</strong></td>
<td>integer</td>
<td>Indicates how to assign colors to <code>zArray</code> data values that do not exactly match the data values in <code>colorMapArray</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>interpPixels</strong></td>
<td>integer</td>
<td>Indicates how pixels between the pixels assigned to the <code>zArray</code> values are colored.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for the plot. This handle can be passed to SetPlotAttribute, GetPlotAttribute, or DeleteGraphPlot. Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>

Parameter Discussion

**zArray** must be one of the following data types (specified in **zDataType**):

```c
VAL_DOUBLE
VAL_FLOAT
VAL_INTEGER
VAL_SHORT_INTEGER
VAL_CHAR
VAL_UNSIGNED_INTEGER
VAL_UNSIGNED_SHORT_INTEGER
VAL_UNSIGNED_CHAR
```

The locations at which the colors are shown on the graph depend on the location of the data values in **zArray**.

- **zArray** should be a two dimensional array of the following form.

  ```c
  zArray[numberOfYPoints][numberOfXPoints]
  ```

- Each element of the array is associated with a pixel on the graph. The pixel associated with element **zArray[y][x]** is located at {x, y} on the graph.

**colorMapArray** contains up to 255 **ColorMapEntry** structures, which consist of:

```c
union {
  char valChar;
  int valInt;
  short valShort;
  float valFloat;
  double valDouble;
  unsigned char valUChar;
  unsigned long valULong;
  unsigned short valUShort;
} dataValue;
int color; /* RGB value */
```

The Color Map array defines how data values in **zArray** are translated into color values. If a data value matches exactly to a data value in one of the **ColorMapEntry** structures, then it is converted to the corresponding color. Otherwise, the following conditions apply.

- If **interpColors** is zero, the color associated with the next higher data value is used.

- If **interpColors** is nonzero, the color is computed using a weighted mean of the colors associated with the Color Map data values immediately above and below the **zArray** value.
• Regardless of the value of interpColors, the following conditions apply.
  – Data values below the lowest Color Map data value are assigned the color of the lowest Color Map data value.
  – Data values above the highest Color Map data value are assigned the value of the hiColor parameter.

If interpColors is nonzero, the numberOfColors must be greater than or equal to 2.

The colorMap entries do not need to be in sorted order.

interpPixels indicates how pixels between the pixels assigned to the zArray values are colored. If interpPixels is zero, an unassigned pixel is given the same color as the closest assigned pixel. If interpPixels is nonzero, an unassigned pixel is first given a data value using a weighted mean of the data values associated with the four closest assigned pixels. Then the color is calculated using the colorMap.

Performance Considerations

If interpPixels is zero, the performance degrades as the number of data points in zArray increases.

If interpPixels is nonzero, the performance degrades as total number of pixels in the plot area increases.

See Also

PlotScaledIntensity

______________________________

PlotLine

int plotHandle = PlotLine (int panelHandle, int controlID, double x1, double y1,
  double x2, double y2, int color);

Purpose

Plots a line into a graph control.

The starting and ending points of the line are specified in terms of X and Y coordinates of the graph.
### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td><strong>x1</strong></td>
<td>double-precision</td>
<td>The horizontal coordinate of the starting point of the line.</td>
</tr>
<tr>
<td></td>
<td><strong>y1</strong></td>
<td>double-precision</td>
<td>The vertical coordinate of the starting point of the line.</td>
</tr>
<tr>
<td></td>
<td><strong>x2</strong></td>
<td>double-precision</td>
<td>The horizontal coordinate of the ending point of the line.</td>
</tr>
<tr>
<td></td>
<td><strong>y2</strong></td>
<td>double-precision</td>
<td>The vertical coordinate of the ending point of the line.</td>
</tr>
<tr>
<td></td>
<td><strong>color</strong></td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
</tbody>
</table>

### Return Value

| **plotHandle**   | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute. |

---

**PlotOval**

```c
int plotHandle = PlotOval(int panelHandle, int controlID, double x1, double y1, double x2, double y2, int color, int fillColor);
```

**Purpose**

Plots a oval into a graph control.

The oval is defined by specifying two opposing corners of a rectangle that enclose the oval in terms of X and Y coordinates of the graph.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>x1</td>
<td>double-precision</td>
<td>The horizontal coordinate of one corner of the rectangle that encloses the oval.</td>
</tr>
<tr>
<td></td>
<td>y1</td>
<td>double-precision</td>
<td>The vertical coordinate of one corner of the rectangle that encloses the oval.</td>
</tr>
<tr>
<td></td>
<td>x2</td>
<td>double-precision</td>
<td>The horizontal coordinate of the opposing corner of the rectangle that encloses the oval.</td>
</tr>
<tr>
<td></td>
<td>y2</td>
<td>double-precision</td>
<td>The vertical coordinate of the opposing corner of the rectangle that encloses the oval.</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
<tr>
<td></td>
<td>fillColor</td>
<td>integer</td>
<td>Specifies the fill color of the figure.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute.</td>
</tr>
</tbody>
</table>

PlotPoint

```c
int plotHandle = PlotPoint(int panelHandle, int controlID, double xCoordinate, double yCoordinate, int pointStyle, int color);
```

Purpose
Plots a point into a graph control.

The position of the point is specified in terms of X and Y coordinates of the graph.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>xCoordinate</td>
<td>double-precision</td>
<td>The horizontal position at which to plot the point.</td>
</tr>
<tr>
<td></td>
<td>yCoordinate</td>
<td>double-precision</td>
<td>The vertical coordinate at which to plot the point.</td>
</tr>
<tr>
<td></td>
<td>pointStyle</td>
<td>integer</td>
<td>The point style to be used when plotting the array. Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>integer</td>
<td>The color of the curve to be plotted.</td>
</tr>
</tbody>
</table>

**Return Value**

| plotHandle | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute. |

**PlotPolygon**

```c
int plotHandle = PlotPolygon (int panelHandle, int controlID, void *xArray, void *yArray, int pointsInPolygon, int xDataType, int yDataType, int color, int fillColor);
```

**Purpose**
Plots a polygon into a graph control.

The polygon is defined by a set of connected X-Y points. The last point is automatically connected to the first point to close the polygon.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>xArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the X axis. The data type must be of the type specified by xDataType.</td>
</tr>
<tr>
<td></td>
<td>yArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by yDataType.</td>
</tr>
<tr>
<td></td>
<td>pointsInPolygon</td>
<td>integer</td>
<td>The number of connected points (corners) in the polygon. There must be at least three corners. pointsInPolygon controls the number of corners plotted, even if the number of elements in the X and Y Arrays are greater than the value of pointsInPolygon.</td>
</tr>
<tr>
<td></td>
<td>xDataType</td>
<td>integer</td>
<td>Specifies the data type of the X array. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td></td>
<td>yDataType</td>
<td>integer</td>
<td>Specifies the data type of the Y array. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
<tr>
<td></td>
<td>fillColor</td>
<td>integer</td>
<td>Specifies the fill color of the figure.</td>
</tr>
</tbody>
</table>

**Return Value**

| plotHandle | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute |
PlotRectangle

```c
int plotHandle = PlotRectangle(int panelHandle, int controlID, double x1, double y1,
                                double x2, double y2, int color, int fillColor);
```

Purpose

Plots a rectangle into a graph control.

Two opposing corners of the rectangle are specified in terms of X and Y coordinates of the graph.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td>x1</td>
<td>double-precision</td>
<td>The horizontal coordinate of one corner of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>y1</td>
<td>double-precision</td>
<td>The vertical coordinate of one corner of the rectangle</td>
</tr>
<tr>
<td></td>
<td>x2</td>
<td>double-precision</td>
<td>The horizontal coordinate of the opposing corner of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>y2</td>
<td>double-precision</td>
<td>The vertical coordinate of the opposing corner of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
<tr>
<td></td>
<td>fillColor</td>
<td>integer</td>
<td>Specifies the fill color of the figure.</td>
</tr>
</tbody>
</table>
Return Value

| plotHandle | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute. |
PlotScaledIntensity

```c
int status = PlotScaledIntensity(int panelHandle, int controlID, void *zArray,
int numberOfXPoints, int numberOfYPoints,
int zDataType, double yGain, double yOffset,
double xGain, double xOffset,
ColorMapEntry colorMapArray[], int hiColor,
int numberOfColors, int interpColors,
int interpPixels);
```

**Purpose**

This function draws a solid rectangular plot in a graph control. It is the same as
PlotIntensity, except that you can apply scaling factors and offsets to the data values.

The plot consists of pixels whose colors correspond to the magnitude of data values in a two-
dimensional array and whose coordinates correspond to the locations of the same data values in
the array, scaled by xGain and yGain and offset by xOffset and yOffset. For instance the pixel
associated with zArray[2][3] is located at the following coordinates.

\[
x = 3 \times xGain + xOffset, \quad y = 2 \times yGain + yOffset
\]

The lower left corner of the plot area is located at

\[
(xOffset, yOffset)
\]

The upper right corner of the plot area is located at

\[
(X-1) \times xGain + xOffset, \quad (Y-1) \times yGain + yOffset
\]

where

X = Number of X Points
Y = Number of Y Points
## Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>panelHandle</strong></td>
<td>integer</td>
<td>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</td>
</tr>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td><strong>zArray</strong></td>
<td>numeric array</td>
<td>An array that contains the data values that are to be converted to colors.</td>
</tr>
<tr>
<td></td>
<td><strong>numberOfXPoints</strong></td>
<td>integer</td>
<td>The number of points to be displayed along the x-axis in each row.</td>
</tr>
<tr>
<td></td>
<td><strong>numberOfYPoints</strong></td>
<td>integer</td>
<td>The number of points to be displayed along the y-axis in each column.</td>
</tr>
<tr>
<td></td>
<td><strong>zDataType</strong></td>
<td>integer</td>
<td>Specifies the data type of the elements in <code>zArray</code>, as well as the data type of the Color Map values.</td>
</tr>
<tr>
<td></td>
<td><strong>yGain</strong></td>
<td>double-precision</td>
<td>Specifies the scaling factor to be applied to the vertical locations implied by <code>zArray</code> vertical index values.</td>
</tr>
<tr>
<td></td>
<td><strong>yOffset</strong></td>
<td>double-precision</td>
<td>Specifies the offset to be added to the vertical locations implied by <code>zArray</code> vertical index values.</td>
</tr>
<tr>
<td></td>
<td><strong>xGain</strong></td>
<td>double-precision</td>
<td>Specifies the scaling factor to be applied to the horizontal locations implied by <code>zArray</code> horizontal index values.</td>
</tr>
<tr>
<td></td>
<td><strong>xOffset</strong></td>
<td>double-precision</td>
<td>Specifies the offset to be added to the horizontal locations implied by <code>zArray</code> horizontal index values.</td>
</tr>
<tr>
<td></td>
<td><strong>colorMapArray</strong></td>
<td>ColorMapEntry</td>
<td>An array of <code>ColorMapEntry</code> structures which specifies how the data values in <code>zArray</code> are translated. The maximum number of entries is 255.</td>
</tr>
<tr>
<td></td>
<td><strong>hiColor</strong></td>
<td>integer</td>
<td>The RGB value to which all <code>zArray</code> values that are higher than the highest data value in <code>colorMapArray</code> are translated.</td>
</tr>
<tr>
<td></td>
<td><strong>numberOfColors</strong></td>
<td>integer</td>
<td>The number of entries in <code>colorMapArray</code>. Must be less than or equal to 255.</td>
</tr>
<tr>
<td></td>
<td><strong>interpColors</strong></td>
<td>integer</td>
<td>Indicates how to assign colors to <code>zArray</code> data values that do not exactly match the data values in the <code>colorMapArray</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>interpPixels</strong></td>
<td>integer</td>
<td>Indicates how pixels between the pixels assigned to the <code>zArray</code> values are colored.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

Parameter Discussion

**zArray** must be one of the following data types (specified in **zDataType**).

```c
VAL_DOUBLE
VAL_FLOAT
VAL_INTEGER
VAL_SHORT_INTEGER
VAL_CHAR
VAL_UNSIGNED_INTEGER
VAL_UNSIGNED_SHORT_INTEGER
VAL_UNSIGNED_CHAR
```

The locations at which the colors are shown on the graph depend on that location of the data values in **zArray**.

- **zArray** should be a two-dimensional array of the following form.
  ```c
  zArray[numberOfYPoints][numberOfXPoints]
  ```
- Each element of the array is associated with a pixel on the graph. The pixel associated with element \( zArray[y][x] \) is located at \( \{ x \times xGain + xOffset, y \times yGain + yOffset \} \) on the graph.

**colorMapArray** contains up to 255 **ColorMapEntry** structures, which consist of:

```c
union {
  char valChar;
  int valInt;
  short valShort;
  float valFloat;
  double valDouble;
  unsigned char valUChar;
  unsigned long valULong;
  unsigned short valUShort;
} data Value;
int color; /* RGB value */
```

The Color Map array defines how data values in **zArray** are translated into color values. If a data value matches exactly to a data value in one of the **ColorMapEntry** structures, then it is converted to the corresponding color. Otherwise, the following conditions apply.

- If **interpColors** is zero, the color associated with the next higher data value is used.
- If **interpColors** is nonzero, the color is computed using a weighted mean of the colors associated with the Color Map data values immediately above and below the **zArray** value.
- Regardless of the value of **interpColors**, the following conditions apply.
– Data below the lowest Color Map data value are assigned the color of the lowest Color Map data value.
– Data values above the highest Color Map data value are assigned the value of the hiColor parameter.

If interpColors is nonzero, the numberOfColors must be greater than or equal to 2.

The colorMapArray entries do not need to be in sorted order.

interpPixels indicates how pixels between the pixels assigned to the zArray values are colored. If interpPixels is zero, an unassigned pixel is given the same color as the closest assigned pixel. If interpPixels is nonzero, an unassigned pixel is first given a data value using a weighted mean of the data values associated with the four closest assigned pixels. Then the color is calculated using the Color Map.

Performance Considerations

If interpPixels is zero, the performance depends on the number of data points in zArray.

If interpPixels is nonzero, the performance depends on the total number of pixels in the plot area.

---

PlotStripChart

```c
int status = PlotStripChart (int panelHandle, int controlID, void *yArray,
                           int numberOfPoints, int startIndex, int skipCount,
                           int yDataType);
```

Purpose

Adds one or more points to each trace in a strip chart control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>yArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by yDataType.</td>
</tr>
<tr>
<td></td>
<td>numberOfPoints</td>
<td>integer</td>
<td>The number of yArray points to add to the strip chart. This value must be an even multiple of the number traces in the strip chart. This value controls the number of points to plot even if the number of elements in yArray is greater than the numberOfPoints.</td>
</tr>
<tr>
<td></td>
<td>startingIndex</td>
<td>integer</td>
<td>The index of the element in the yArray where the first block of data begins. This value is zero-based and must be an even multiple of the number of traces in the strip chart. The default value is 0.</td>
</tr>
<tr>
<td></td>
<td>skipCount</td>
<td>integer</td>
<td>The number of yArray elements to skip over (increment) after each set of points is plotted. The default is 0.</td>
</tr>
<tr>
<td></td>
<td>yDataType</td>
<td>integer</td>
<td>Specifies the data type of the yArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

Parameter Discussion

numberOfPoints

The right edge of the strip chart is considered a grid line. The Points per Screen attribute of the strip chart encompasses this grid line at the right edge of the strip chart. To synchronize your data with the grid lines, you should set the strip chart Points per Screen attribute to one greater than the datapoints per screen, in other words,

strip chart Points per Screen = datapoints per screen + 1.
You can set the **Points per Screen** attribute of the strip chart in the User Interface Editor or you can use `SetCtrlAttribute` function with the `ATTR_POINTS_PER_SCREEN` attribute.

The datapoints per screen is controlled by the **numberofPoints** parameter of `PlotStripChart`.

For Continuous mode strip charts, if you do not follow this algorithm, you will notice the grid lines drifting away from your datapoints.

For Sweep & Block mode, if you do not follow this algorithm, you will notice gaps on the right side of the strip chart, as described in the following example.

If you set Points per Screen to 100 points, the first call to `PlotStripChart` (with 100 points) plots from 0 to 99. The second call to `PlotStripChart` plots from 99 to 198 and then plots the last point (199) at the left edge of the strip chart. If the second `PlotStripChart` starts at 100—as you might expect—the information between 99 and 100 never appears on the screen.

To make all data appear on the screen, set your strip chart Points per Screen to 101 points.

**skipCount**

Suppose you have an array that contains 4 data sets (A,B,C,D) and you want to plot each A-B pair into a strip chart control. If the elements are ordered in the following manner,

A B C D A B C D A B C D

then entering a **skipCount** of 2 will cause each C-D pair to be skipped before the next A-B pair is plotted.

---

**PlotStripChartPoint**

```c
int status = PlotStripChartPoint (int panelHandle, int controlID, double y);
```

**Purpose**

This function provides a simple interface for adding one point to a strip chart that contains only one trace. The same operation can be performed using `PlotStripChart`.

Refer to the help text for the `PlotStripChart` function if you need more information about strip chart plots.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td><strong>y</strong></td>
<td>double-precision</td>
<td>The data value to be plotted along the y-axis.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

---

**PlotText**

```c
int plotHandle = PlotText (int panelHandle, int controlID, double xCoordinate, double yCoordinate, char text[], int font, int textColor, int backgroundColor);
```

Purpose

Plots a text string into a graph control.

The origin of the text is the lower left corner of the string.

The origin of the text is specified in terms of the X and Y coordinates of the graph. If the graph attribute ATTR_SHIFT_TEXT_PLOTS is non-zero and the text origin is within the graph area, the text shifts to the left and/or down until it is completely visible. If ATTR_SHIFT_TEXT_PLOTS is zero, the text does not shift.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>xCoordinate</td>
<td>double-precision string</td>
<td>The horizontal position at which to place the left edge of the text within the graph.</td>
</tr>
<tr>
<td></td>
<td>yCoordinate</td>
<td>double-precision string</td>
<td>The vertical position at which to place the bottom edge of the text within the graph.</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>string</td>
<td>The string to be plotted.</td>
</tr>
<tr>
<td></td>
<td>font</td>
<td>integer</td>
<td>Selects the text font. See Table 3-5 in Chapter 3 for valid fonts.</td>
</tr>
<tr>
<td></td>
<td>textColor</td>
<td>integer</td>
<td>Specifies the color of the plotted text. Table 3-3 in Chapter 3 presents a list of the common colors.</td>
</tr>
<tr>
<td></td>
<td>backgroundColor</td>
<td>integer</td>
<td>Specifies the background color of the plotted text. Table 3-3 in Chapter 3 presents a list of the common colors.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute.</td>
</tr>
</tbody>
</table>
PlotWaveform

```c
int plotHandle = PlotWaveform(int panelHandle, int controlID, void *yArray,
                               int numberOfPoints, int yDataType, double yGain,
                               double yOffset, double initialX, double xIncrement,
                               int plotStyle, int pointStyle, int lineStyle,
                               int pointFrequency, int color);
```

**Purpose**

Plots a waveform into a graph control.

The values in Y Array are scaled according to Y Gain and Y Offset. The X axis timebase is scaled according to Initial X and X Increment. Each point in the plot is computed as follows:

\[
x_i = (i \times XInc) + InitX
\]

\[
y_i = (wfm_i \times YGain) + YOff
\]

where \( i \) is the index of the point in the waveform array.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>PanelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>yArray</td>
<td>void *</td>
<td></td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by yDataType.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td></td>
<td>The number of points to be plotted.</td>
</tr>
<tr>
<td>yDataType</td>
<td>integer</td>
<td></td>
<td>Specifies the data type of yArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td>yGain</td>
<td>double-precision</td>
<td></td>
<td>Specifies the gain to be applied to the waveform (yArray) data.</td>
</tr>
</tbody>
</table>

(continues)
Parameters (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yOffset</td>
<td>double-precision</td>
<td>Specifies a constant offset to be added to the waveform (yArray ) data. The default value is 0.0</td>
</tr>
<tr>
<td>initialX</td>
<td>double-precision</td>
<td>Specifies the initial value for the X axis.</td>
</tr>
<tr>
<td>xIncrement</td>
<td>double-precision</td>
<td>Specifies the increment along the X axis for each new point.</td>
</tr>
<tr>
<td>plotStyle</td>
<td>integer</td>
<td>The curve style to be used when plotting the data points. See Table 3-24 in Chapter 3 for a list of plot styles.</td>
</tr>
<tr>
<td>pointStyle</td>
<td>integer</td>
<td>The point style to be used when plotting the array. The point style determines the type of marker drawn when the plot style is VAL_CONNECTED_POINTS or VAL_SCATTER. See Table 3-22 in Chapter 3 for a list of point styles.</td>
</tr>
<tr>
<td>lineStyle</td>
<td>integer</td>
<td>This control selects the line style. See Table 3-23 in Chapter 3 for a list of line styles.</td>
</tr>
<tr>
<td>pointFrequency</td>
<td>integer</td>
<td>Specifies the point interval at which marker symbols are drawn when the curve style is VAL_CONNECTED_POINTS or VAL_SCATTER.</td>
</tr>
<tr>
<td>color</td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute.</td>
</tr>
</tbody>
</table>
PlotX

```
int plotHandle = PlotX (int panelHandle, int controlID, void *xArray,
               int numberOfPoints, int xDataType, int plotStyle,
               int pointStyle, int linestyle, int pointFrequency, int color);
```

**Purpose**

Plots an array of X values against its indices along the Y axis. The plot is displayed in a graph control on the specified panel.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>xArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the X axis. The data type must be of the type specified by xDataType.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in xArray is greater than the numberOfPoints.</td>
</tr>
<tr>
<td>xDataType</td>
<td>integer</td>
<td>Specifies the data type of the xArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td>plotStyle</td>
<td>integer</td>
<td>The curve style to be used when plotting the data points. See Table 3-24 in Chapter 3 for a list of plot styles.</td>
</tr>
<tr>
<td>pointStyle</td>
<td>integer</td>
<td>The point style to be used when plotting the array. The point style determines the type of marker drawn when the plot style is VAL_CONNECTED_POINTS or VAL_SCATTER. See Table 3-22 in Chapter 3 for a list of point styles.</td>
</tr>
<tr>
<td>linestyle</td>
<td>integer</td>
<td>This control selects the line style. See Table 3-23 in Chapter 3 for a list of line styles.</td>
</tr>
<tr>
<td>pointFrequency</td>
<td>integer</td>
<td>Specifies the point interval at which marker symbols are drawn when the curve style is VAL_CONNECTED_POINTS or VAL_SCATTER.</td>
</tr>
<tr>
<td>color</td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
</tbody>
</table>
Return Value

| plotHandle | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute. |

PlotXY

```c
int plotHandle = PlotXY (int panelHandle, int controlID, void *xArray, void *yArray, int numberofPoints, int xDataType, int yDataType, int plotStyle, int pointStyle, int lineStyle, int pointFrequency, int color);
```

Purpose

Plots an array of X values against an array of Y values.

The plot is displayed in a graph control on the specified panel.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>xArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the X axis. The data type must be of the type specified by xDataType.</td>
</tr>
<tr>
<td></td>
<td>yArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by yDataType.</td>
</tr>
<tr>
<td></td>
<td>numberofPoints</td>
<td>integer</td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in X Array is greater than the numberofPoints.</td>
</tr>
</tbody>
</table>
### Parameters (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xDataType</td>
<td>integer</td>
<td>Specifies the data type of the xArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td>yDataType</td>
<td>integer</td>
<td>Specifies the data type of the yArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td>plotStyle</td>
<td>integer</td>
<td>The curve style to be used when plotting the data points. See Table 3-24 in Chapter 3 for a list of plot styles.</td>
</tr>
<tr>
<td>pointStyle</td>
<td>integer</td>
<td>The point style to be used when plotting the array. The point style determines the type of marker drawn when the plot style is VAL_CONNECTED_POINTS or VAL_SCATTER. See Table 3-22 in Chapter 3 for a list of point styles.</td>
</tr>
<tr>
<td>lineStyle</td>
<td>integer</td>
<td>This control selects the line style. See Table 3-23 in Chapter 3 for a list of line styles.</td>
</tr>
<tr>
<td>pointFrequency</td>
<td>integer</td>
<td>Specifies the point interval at which marker symbols are drawn when the curve style is VAL_CONNECTED_POINTS or VAL_SCATTER</td>
</tr>
<tr>
<td>color</td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format 0x00RRGGBB. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute.</td>
</tr>
</tbody>
</table>

### PlotY

```c
int plotHandle = PlotY (int panelHandle, int controlID, void *yArray,
                        int numberOfPoints, int yDataType, int plotStyle,
                        int pointStyle, int lineStyle, int pointFrequency, int color);
```

### Purpose

Plots an array of Y values against its indices along the X axis. The plot is displayed in a graph control on the specified panel.
## Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td><strong>yArray</strong></td>
<td>void *</td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by <code>yDataType</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>numberofPoints</strong></td>
<td>integer</td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in <code>xArray</code> is greater than the <code>numberofPoints</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>yDataType</strong></td>
<td>integer</td>
<td>Specifies the data type of the <code>yArray</code>. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td></td>
<td><strong>plotStyle</strong></td>
<td>integer</td>
<td>The curve style to be used when plotting the data points. See Table 3-24 in Chapter 3 for a list of plot styles.</td>
</tr>
<tr>
<td></td>
<td><strong>pointStyle</strong></td>
<td>integer</td>
<td>The point style to be used when plotting the array. The point style determines the type of marker drawn when the plot style is <code>VAL_CONNECTED_POINTS</code> or <code>VAL_SCATTER</code>. See Table 3-22 in Chapter 3 for a list of point styles.</td>
</tr>
<tr>
<td></td>
<td><strong>lineStyle</strong></td>
<td>integer</td>
<td>This control selects the line style. See Table 3-23 in Chapter 3 for a list of line styles.</td>
</tr>
<tr>
<td></td>
<td><strong>pointFrequency</strong></td>
<td>integer</td>
<td>Specifies the point interval at which marker symbols are drawn when the curve style is <code>VAL_CONNECTED_POINTS</code> or <code>VAL_SCATTER</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>color</strong></td>
<td>integer</td>
<td>Specifies the color of the curve to be plotted. An RGB value is a 4-byte integer with the hexadecimal format <code>0x00RRGGBB</code>. RR, GG, and BB are the respective red, green, and blue components of the color value. See Table 3-3 in Chapter 3 for a list of the common colors.</td>
</tr>
</tbody>
</table>
Return Value

| plotHandle         | integer | The handle for the plot. This handle can be passed to DeleteGraphPlot to delete the individual plot. Refer to Appendix A for error codes. It can also be passed to SetPlotAttribute and GetPlotAttribute. |

PointEqual

```c
int pointsAreEqual = PointEqual(Point point1, Point point2);
```

Purpose

Indicates if two points are the same.

Returns 1 if the x and y values of two specified points are the same. Returns 0 otherwise.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>point1</th>
<th>Point</th>
<th>A point structure.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>point2</td>
<td>Point</td>
<td>A point structure.</td>
</tr>
</tbody>
</table>

Return Value

| pointsAreEqual | integer | An indication of the two points are the same. |

Return Codes

| 1         | The x and y coordinates in the two Point structures are the same. |
| 0         | The x and y coordinates in the two Point structures are not the same. |

See Also

MakePoint
**PointPinnedToRect**

```c
void PointPinnedToRect(Point point, Rect rect, Point *pinnedPoint);
```

**Purpose**

This function ensures that a point is within a specified rectangular area. If the point is already enclosed by the rectangle, the location of the point remains unchanged. If the point is outside the rectangle, its location is set to the nearest point on the edge of the rectangle.

The `Point` structure containing the original location is not modified. The calculated location is stored in another `Point` structure.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>point</th>
<th>Point</th>
<th>A <code>Point</code> structure specifying the original location of the point.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rect</td>
<td>Rect</td>
<td>A <code>Rect</code> structure specifying the rectangle to which the point is to be pinned.</td>
</tr>
<tr>
<td>Output</td>
<td>pinnedPoint</td>
<td>Point</td>
<td>The <code>Point</code> structure in which the calculated location is stored.</td>
</tr>
</tbody>
</table>

**Return Value**

None

**See Also**

`MakePoint`, `MakeRect`.

---

**PointSet**

```c
void PointSet(Point *point, int xCoordinate, int yCoordinate);
```

**Purpose**

Sets the values in an existing `Point` structure. The `Point` structure defines the location of a point.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>xCoordinate</th>
<th>integer</th>
<th>The new horizontal location of the point.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yCoordinate</td>
<td>integer</td>
<td>The new vertical location of the point.</td>
</tr>
<tr>
<td>Output</td>
<td>point</td>
<td>Point</td>
<td>The Point structure in which the new values are set.</td>
</tr>
</tbody>
</table>

Return Value

None

See Also

MakePoint

---

**PostDeferredCall**

```c
int status = PostDeferredCall(DeferredCallbackPtr deferredFunction, void *callbackData);
```

Purpose

Posts a function to LabWindows/CVI that is to be called at the next occurrence of `GetUserEvent`, `RunUserInterface`, or `ProcessSystemEvents`.

`PostDeferredCall` would typically be used in a function installed as an asynchronous interrupt handler. The asynchronous interrupt handler is limited in what it can do. It cannot do anything time consuming and it cannot call into the User Interface Library. The function that is posted by `PostDeferredCall` contains the code that cannot be executed at interrupt time.

This is useful when external devices generate interrupts during source program execution.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>deferredFunction</th>
<th>DeferredCallbackPtr</th>
<th>A pointer to the function whose execution will be deferred until the next <code>GetUserEvent</code>, <code>RunUserInterface</code>, or <code>ProcessSystemEvents</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>callbackData</td>
<td>void *</td>
<td>A pointer to user-defined data passed to the deferred function.</td>
</tr>
</tbody>
</table>
Return Value

| status       | integer | Refer to Appendix A for error codes. |

Parameter Discussion

The function pointed to by `deferredFunction` takes the following form:

```c
void CVICALLBACK DeferredCallbackFunction (void *callbackData);
```

---

PrintCtrl

```c
int printStatus = PrintCtrl (int panelHandle, int controlID, char fileName[], int scaling, int confirmDialogBox);
```

Purpose

Prints the selected control.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>fileName</td>
<td>string</td>
<td></td>
<td>The name of the output file. If the name is non-empty, the output will be redirected to the file. If the name is not a complete pathname, the file will be created relative to the current working directory.</td>
</tr>
<tr>
<td>scaling</td>
<td>integer</td>
<td></td>
<td>Selects the scaling mode for printing. 1 expands the object to full size. 0 (zero) prints the object at the same relative location and size as displayed on the screen.</td>
</tr>
<tr>
<td>confirmDialogBox</td>
<td>integer</td>
<td></td>
<td>Displays a dialog box before printing to confirm print attributes. Requested and supported attribute values are shown for the current printer, allowing the user to change attribute values during run-time.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>printStatus</th>
<th>integer</th>
<th>Returns the status of the print operation. Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

Return Codes

The value returned by the PrintCtrl function is a bit-field.

<table>
<thead>
<tr>
<th>VAL_TOO_MANY_COPIES</th>
<th>(1&lt;&lt;0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_NO_MULTIPLE_COPIES</td>
<td>(1&lt;&lt;1)</td>
</tr>
<tr>
<td>VAL_NO_DUPLEX</td>
<td>(1&lt;&lt;2)</td>
</tr>
<tr>
<td>VAL_NO_LANDSCAPE</td>
<td>(1&lt;&lt;3)</td>
</tr>
<tr>
<td>VAL_CANT_FORCE_MONO</td>
<td>(1&lt;&lt;4)</td>
</tr>
<tr>
<td>VAL_NO_SUCH_XRESOLUTION</td>
<td>(1&lt;&lt;5)</td>
</tr>
<tr>
<td>VAL_NO_MULTIPLE_XRESOLUTION</td>
<td>(1&lt;&lt;6)</td>
</tr>
<tr>
<td>VAL_NO_SUCH_YRESOLUTION</td>
<td>(1&lt;&lt;7)</td>
</tr>
<tr>
<td>VAL_NO_MULTIPLE_YRESOLUTION</td>
<td>(1&lt;&lt;8)</td>
</tr>
<tr>
<td>VAL_NO_SEPARATE_YRESOLUTION</td>
<td>(1&lt;&lt;9)</td>
</tr>
<tr>
<td>VAL_USER_CANCEL</td>
<td>(1&lt;&lt;10)</td>
</tr>
</tbody>
</table>

PrintPanel

```c
int printStatus = PrintPanel(int panelHandle, char fileName[], int scaling,
                              int scope, int confirmDialogBox);
```

Purpose

Prints the selected panel.
### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>fileName</strong></td>
<td>string</td>
<td>The name of the output file. If the name is non-empty, the output will be redirected to the file. If the name is not a complete pathname, the file will be created relative to the current working directory.</td>
</tr>
<tr>
<td></td>
<td><strong>scaling</strong></td>
<td>integer</td>
<td>Selects the scaling mode for printing. 1 expands the object to the size of an entire page. 0 (zero) prints the object at the same relative location and size as displayed on the screen.</td>
</tr>
<tr>
<td></td>
<td><strong>scope</strong></td>
<td>integer</td>
<td>Selects the portion of the specified panel to be printed. VAL_VISIBLE_AREA or VAL_FULL_PANEL.</td>
</tr>
<tr>
<td></td>
<td><strong>confirmDialogBox</strong></td>
<td>integer</td>
<td>Displays a dialog box before printing to confirm print attributes. Requested and supported attribute values are shown for the current printer, allowing the user to change attribute values during run-time.</td>
</tr>
</tbody>
</table>

### Return Value

| **printStatus** | integer | Returns the status of the print operation. Refer to Appendix A for error codes. |

### Return Codes

The value returned by the PrintPanel function is a bit-field.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_TOO_MANY_COPIES</td>
<td>(1&lt;&lt;0)</td>
</tr>
<tr>
<td>VAL_NO_MULTIPLE_COPIES</td>
<td>(1&lt;&lt;1)</td>
</tr>
<tr>
<td>VAL_NO_DUPLEX</td>
<td>(1&lt;&lt;2)</td>
</tr>
<tr>
<td>VAL_NO_LANDSCAPE</td>
<td>(1&lt;&lt;3)</td>
</tr>
<tr>
<td>VAL_CANT_FORCE_MONO</td>
<td>(1&lt;&lt;4)</td>
</tr>
<tr>
<td>VAL_NO_SUCH_XRESOLUTION</td>
<td>(1&lt;&lt;5)</td>
</tr>
<tr>
<td>VAL_NO_MULTIPLE_XRESOLUTIONS</td>
<td>(1&lt;&lt;6)</td>
</tr>
<tr>
<td>VAL_NO_SUCH_YRESOLUTION</td>
<td>(1&lt;&lt;7)</td>
</tr>
<tr>
<td>VAL_NO_MULTIPLE_YRESOLUTIONS</td>
<td>(1&lt;&lt;8)</td>
</tr>
<tr>
<td>VAL_NO_SEPARATE_YRESOLUTION</td>
<td>(1&lt;&lt;9)</td>
</tr>
<tr>
<td>VAL_USER_CANCEL</td>
<td>(1&lt;&lt;10)</td>
</tr>
</tbody>
</table>
Parameter Discussion

If VAL_VISIBLE_AREA is the scope, only that portion of the panel visible on the screen is printed. Menu bars, scroll bars, and a frame are printed along with the visible portion. If VAL_FULL_PANEL is the scope, the entire panel is printed. No menu bars, scroll bars, or frames are printed. Regardless of the scope, objects within child panels will be clipped to the frame of the child panel.

Note: By default PrintPanel uses the following settings:
• ATTR_PAPER_WIDTH is set to VAL_USE_PRINTER_DEFAULT
• ATTR_PAPER_HEIGHT is set to VAL_INTEGRAL_SCALE.
If the printout does not fit on the page, call SetPrintAttribute before PrintPanel which changes the settings as follows:
• ATTR_PAPER_WIDTH is set to VAL_INTEGRAL_SCALE
• ATTR_PAPER_HEIGHT is set to VAL_USE_PRINTER_DEFAULT.

PrintTextBuffer

int status = PrintTextBuffer(char buffer[], char outputFile[]);

Purpose

Prints the contents of a text buffer. You may direct the output to the printer or to a file.

Newline/carriage-return characters are honored.

Tabs are expanded according to the current state of the ATTR_TAB_INTERVAL attribute. This attribute can be modified using the SetPrintAttribute function. The default tab interval is 4.

Text that extends beyond the end of the paper can be truncated or wrapped, depending on the state of the ATTR_TEXT_WRAP attribute. This attribute can be modified using the SetPrintAttribute function. The default is 'wrap'.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>buffer</th>
<th>string</th>
<th>The text to be printed. Must be terminated by an ASCII NUL byte.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>outputFile</td>
<td>string</td>
<td>The name of a file to which to direct the output. If &quot; &quot;, the output is directed to the printer.</td>
</tr>
</tbody>
</table>
Return Value

| status     | integer | Refer to Appendix A for error codes. |

Parameter Discussion

If `outputFile` is a not a complete pathname, it is created relative to the current working directory.

See Also

PrintTextFile.

---

PrintTextFile

```c
int status = PrintTextFile(char fileName[], char outputFile[]);
```

Purpose

Prints a text file. You can direct the output to the printer or to a file.

Newline/carriage-return characters are honored.

Tabs are expanded according to the current state of the `ATTR_TAB_INTERVAL` attribute. This attribute can be modified using the `SetPrintAttribute` function. The default tab interval is 4.

Text that extends beyond the end of the paper can be truncated or wrapped, depending on the state of the `ATTR_TEXT_WRAP` attribute. This attribute can be modified using the `SetPrintAttribute` function. The default setting is 'wrap'.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>fileName</th>
<th>string</th>
<th>The name of the file to print.</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputFile</td>
<td>string</td>
<td></td>
<td>The name of a file to which to direct the output. If &quot;&quot;, the output is directed to the printer.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

Parameter Discussion

`fileName` may be a complete pathname or a simple filename. If the name is a simple filename (in other words, contains no directory path), then the following conditions apply.
• If the file is listed in the project, the file is located using the pathname from the project.

• Otherwise, the file is located in the directory containing the project.

If **outputFile** is a not a complete pathname, it is created relative to the current working directory

**See Also**

PrintTextBuffer.

---

**ProcessDrawEvents**

```c
int status = ProcessDrawEvents(void);
```

**Purpose**

While inside of a callback function or in code that does not call RunUserInterface or GetUserEvent, the user interface is not updated. If a particular function is overly time-consuming, it essentially “locks out” user interface updates. To force these updates to be processed, call ProcessDrawEvents when you want the user interface to be updated.

**Note:** The user interface is updated automatically by the GetUserEvent function and when a callback returns.

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

---

**ProcessSystemEvents**

```c
int status = ProcessSystemEvents(void);
```

**Purpose**

While inside of a callback function or in code that does not call RunUserInterface or GetUserEvent, user interface and system events are not processed. If a particular function is overly time-consuming, it will essentially “lock out” user interface and system events. To force these events to be processed, call ProcessSystemEvents occasionally in the code that is locking out system events. Care must be taken when using this function, because it can allow other callback functions to be executed before ProcessSystemEvents completes.

This function processes all pending system events. These include:
- System events that have been delayed or suspended by a user application. For example: keystrokes, mouse events, and screen updates.

- Events generated by other applications. For example: Windows messages intended to invoke a callback installed with RegisterWinMsgCallback.

Note: This function is called automatically by the GetUserEvent function and after a callback returns.

---

**PromptPopup**

```c
int status = PromptPopup (char title[], char message[], char responseBuffer[],
                          int maxResponseLength);
```

**Purpose**

Displays a prompt message in a dialog box and waits for the user to type a reply.

**Parameters**

| Input     | title       | string | The title to be displayed on the dialog box. |
|           | message     | string | The message to be displayed on the dialog box. |
|           | maxResponseLength | integer | The \n character can be used to create multi-line messages. |
|           |             |        | The maximum number of bytes the user is allowed to enter. The responseBuffer must be large enough to contain all of the user’s input plus one ASCII NULL byte. |
| Output    | responseBuffer | string | The buffer in which the user’s response is stored. The buffer must be large enough to hold maxResponseLength bytes plus one ASCII NULL byte. |

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

---

**QueueUserEvent**

```c
int status = QueueUserEvent (int eventNumber, int panelHandle, int controlID);
```
Purpose

QueueUserEvent is used to place a programmer-defined event in the GetUserEvent queue.

Event numbers 1000 to 10000 are reserved for programmer-defined events. Programmer-defined events are returned by GetUserEvent.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>eventNumber</th>
<th>integer</th>
<th>Is an integer value between 1000 and 10000 placed in the event queue which can later be processed by GetUserEvent.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. Pass 0 (zero) when the event doesn’t apply to a particular panel.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function. Pass 0 (zero) when the event doesn’t apply to a particular control.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

QuitUserInterface

```c
int status = QuitUserInterface (int returnCode);
```

Purpose

QuitUserInterface should be called only from within a callback function invoked during execution of RunUserInterface. It causes RunUserInterface to return.
The value returned by `RunUserInterface` is the value that was passed to `QuitUserInterface`.

**Parameters**

| Input | `returnCode` | integer | The value that will be returned from the call to `RunUserInterface` that is being terminated. This parameter can be used as a flag to pass information back through the `RunUserInterface` function. The value should be greater than or equal to zero. |

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

---

**RecallPanelState**

```c
int status = RecallPanelState(int panelHandle, char filename[], int stateIndex);
```

**Purpose**

Reads a panel state from a file created with the `SavePanelState` function. If the panel is currently visible, all controls are updated to reflect their new states.

**Note:** *If the panel has been modified in the User Interface Editor or programmatically since the panel state was saved, recalling the panel state may fail or may result in a modified panel state.*

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>panelHandle</code></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>filename</code></td>
<td>string</td>
<td>The name of the file in which the panel state was saved. If the name is a simple filename (in other words, contains no directory path), then the file is loaded from the directory containing the project.</td>
</tr>
<tr>
<td></td>
<td><code>stateIndex</code></td>
<td>integer</td>
<td>The same state index assigned to the panel state when it was saved using <code>SavePanelState</code>.</td>
</tr>
</tbody>
</table>
Return Value

| status | integer | Refer to Appendix A for error codes. |

---

**RectBottom**

```c
int  bottom = RectBottom( Rect  rect);
```

**Purpose**

Returns the y coordinate of the bottom edge a rectangle. The bottom edge is **not** enclosed by the rectangle. It is computed as follows.

`bottom = rect.top + rect.height`

**Parameter**

| Input | rect  | Rect | Specifies the location and size of a rectangle. |

**Return Value**

| bottom | integer | The y coordinate of the bottom of the rectangle. The bottom is not enclosed by the rectangle, and is equal to `rect.top + rect.height`. |

**See Also**

RectRight

---

**RectCenter**

```c
void RectCenter( Rect  rect, Point  *center);
```

**Purpose**

Calculates the location of the center point of the specified rectangle. For even heights (or widths), the center point is rounded towards the top (or left).
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>rect</th>
<th>Rect</th>
<th>Specifies the location and size of a rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>center</td>
<td>Point</td>
<td>Specifies the location of the center of the rectangle.</td>
</tr>
</tbody>
</table>

Return Value

None

RectContainsPoint

```c
int containsPoint = RectContainsPoint(Rect rect, Point point);
```

Purpose

Returns 1 if the specified rectangle encloses the specified point. Returns 0 otherwise. The rectangle is considered to enclose the point if the point is in the interior of the rectangle or on its frame.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>rect</th>
<th>Rect</th>
<th>Specifies the location and size of a rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>point</td>
<td>Point</td>
<td>Specifies the location of the center of the rectangle.</td>
</tr>
</tbody>
</table>

Return Value

| containsPoint | integer | Indicates if rect contains point. |

Return Codes

<table>
<thead>
<tr>
<th>value</th>
<th>point is in the interior or on the frame of the rectangle specified by rect.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>point is in the interior or on the frame of the rectangle specified by rect.</td>
</tr>
<tr>
<td>0</td>
<td>point is outside the frame of the rectangle specified by rect.</td>
</tr>
</tbody>
</table>
RectContainsRect

```c
int containsRect = RectContainsRect (Rect rect1, Rect rect2);
```

**Purpose**

Returns 1 if the first rectangle encloses the second rectangle. Returns 0 otherwise. A rectangle is considered to enclose another rectangle if every point of the second rectangle is in the interior or on the frame of the first rectangle. (A rectangle encloses itself.)

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>rect1</th>
<th>Rect</th>
<th>Specifies the location and size of a rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rect2</td>
<td>Rect</td>
<td>Specifies the location and size of a rectangle.</td>
</tr>
</tbody>
</table>

**Return Value**

| containsRect | integer | Indicates if rect1 encloses rect2. |

**Return Codes**

| 1 | rect1 encloses rect2. |
| 0 | rect1 does not enclose rect2. |

RectEmpty

```c
int isEmpty = RectEmpty (Rect rect);
```

**Purpose**

Returns 1 if the specified rectangle is empty. Returns 0 otherwise. A rectangle is considered to be empty if either its height or width is less than or equal to zero.

**Parameter**

| Input | rect | Rect | Specifies the location and size of a rectangle. A Rect structure. |

**Return Value**

| isEmpty | integer | Indicates if the rectangle specified by rect is empty. |
Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Either <code>rect.height</code> or <code>rect.width</code> of is less than or equal to zero.</td>
</tr>
<tr>
<td>0</td>
<td>Both <code>rect.height</code> and <code>rect.width</code> are greater than zero.</td>
</tr>
</tbody>
</table>

RectEqual

```c
int areEqual = RectEqual (Rect rect1, Rect rect2);
```

Purpose

Returns 1 if the location and size of the two specified rectangles are identical. Returns 0 otherwise.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><code>rect1</code></th>
<th>Rect</th>
<th>Specifies the location and size of a rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rect2</code></td>
<td>Rect</td>
<td>Rect</td>
<td>Specifies the location and size of a rectangle.</td>
</tr>
</tbody>
</table>

Return Value

| `areEqual` | integer | Indicates if the top, left, height, and width values in `rect1` are identical to those of `rect2`. |

Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>rect1</code> and <code>rect2</code> have the identical top, left, height, and width values.</td>
</tr>
<tr>
<td>0</td>
<td><code>rect1</code> and <code>rect2</code> do not have the identical top, left, height, and width values.</td>
</tr>
</tbody>
</table>

RectGrow

```c
void RectGrow (Rect *rect, int dx, int dy);
```

Purpose

Modifies the values in a `Rect` structure so that the rectangle it defines grows or shrinks around its current center point.
Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>rect</th>
<th>Rect</th>
</tr>
</thead>
<tbody>
<tr>
<td>On input, specifies the size and location of a rectangle. On output, specifies a rectangle of a different size but the same center point.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>dx</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>On input, specifies the size and location of a rectangle. On output, specifies a rectangle of a different size but the same center point.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>dy</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>On input, specifies the size and location of a rectangle. On output, specifies a rectangle of a different size but the same center point.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

None

RectIntersection

```c
int rectsIntersect = RectIntersection(Rect rect1, Rect rect2, Rect *intersectionRect);
```

Purpose

Returns an indication of whether two rectangles are intersecting. If they are, the function fills in a Rect structure describing the intersection area.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>rect1</th>
<th>Rect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the location and size of a rectangle.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>rect2</th>
<th>Rect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the location and size of a rectangle.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>intersectionRect</th>
<th>Rect</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Rect structure set to the largest rectangle enclosed by both rect1 and rect2. If rect1 and rect2 do not intersect, this parameter is set to an empty rectangle (height and width of zero). You may pass 0 for this parameter.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>rectsIntersect</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates if rect1 and rect2 intersect (have any points in common).</td>
<td></td>
</tr>
</tbody>
</table>
Return Codes

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rect1 and rect2 intersect.</td>
</tr>
<tr>
<td>0</td>
<td>rect1 and rect2 do not intersect.</td>
</tr>
</tbody>
</table>

RectMove

```c
void RectMove (Rect *rect, Point point);
```

Purpose

Modifies a Rect structure so that the top, left corner of the rectangle it defines is at the specified point.

Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/Output</td>
<td>rect</td>
<td>Rect On input, specifies the size and location of a rectangle. On output, specifies a rectangle of the same size but a different location.</td>
</tr>
<tr>
<td>Input</td>
<td>point</td>
<td>Point A Point structure specifying the new location of the top, left corner of the rectangle.</td>
</tr>
</tbody>
</table>

Return Value

None

RectOffset

```c
void RectOffset (Rect *rect, int dx, int dy);
```

Purpose

Modifies the values in a Rect structure to shift the location of the rectangle it defines.
Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>rect</th>
<th>Rect</th>
<th>On input, specifies the size and location of a rectangle. On output, specifies a rectangle of the same size but a different location.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>dx</td>
<td>integer</td>
<td>The amount to shift the rectangle horizontally. Use a positive value to shift the rectangle to the right. Use a negative value to shift the rectangle to the left.</td>
</tr>
<tr>
<td></td>
<td>dy</td>
<td>integer</td>
<td>The amount to shift the rectangle vertically. Use a positive value to shift the rectangle down. Use a negative value to shift the rectangle up.</td>
</tr>
</tbody>
</table>

Return Value

None

RectRight

```c
int rightEdge = RectRight (Rect rect);
```

Purpose

Returns the x coordinate of the right edge a rectangle. The right edge is not enclosed by the rectangle. It is computed as follows.

```
rightEdge = rect.left + rect.width
```

Parameter

<table>
<thead>
<tr>
<th>Input</th>
<th>rect</th>
<th>Rect</th>
<th>Specifies a rectangle.</th>
</tr>
</thead>
</table>

Return Value

<table>
<thead>
<tr>
<th>rightEdge</th>
<th>integer</th>
<th>The x coordinate of the right edge of the rectangle. The right edge is not enclosed by the rectangle, and is equal to rect.left + rect.width.</th>
</tr>
</thead>
</table>

See Also

RectBottom
**RectSameSize**

```c
int areSameSize = RectSameSize(Rect rect1, Rect rect2);
```

**Purpose**

Returns 1 if the two specified rectangle have the same height and width. Returns 0 otherwise.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>rect1</th>
<th>Rect</th>
<th>Specifies the location and size of a rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rect2</td>
<td>Rect</td>
<td>Specifies the location and size of a rectangle.</td>
</tr>
</tbody>
</table>

**Return Value**

| areSameSize | integer | Indicates if the height, and width values in rect1 are identical to those of rect2. |

**Return Codes**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rect1 and rect2 have the identical height, and width.</td>
</tr>
<tr>
<td>0</td>
<td>rect1 and rect2 do not have the identical height and width.</td>
</tr>
</tbody>
</table>

---

**RectSet**

```c
void RectSet(Rect *rect, int top, int left, int height, int width);
```

**Purpose**

Sets the values in an existing Rect structure. The Rect structure defines the location and size of a rectangle.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>top</th>
<th>integer</th>
<th>The new location of the top edge of the rectangle.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>left</td>
<td>integer</td>
<td>The new location of the left edge of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>integer</td>
<td>The new height of the rectangle.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>The new width of the rectangle.</td>
</tr>
</tbody>
</table>

| Output | rect | Rect | The Rect structure in which the new values are set. |

Return Value

None

See Also

MakeRect

---

RectSetBottom

```c
void RectSetBottom(Rect *rect, int bottom);
```

Purpose

Sets the height of a `Rect` structure so that the bottom edge of the rectangle it defines is at the specified location. The bottom edge of the rectangle is **not** enclosed by the rectangle and is equal to the top plus the height.

Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>rect</th>
<th>Rect</th>
<th>On input, specifies the size and location of a rectangle. On output, specifies the same rectangle except with a different bottom edge.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>bottom</td>
<td>integer</td>
<td>The y coordinate of the new bottom edge.</td>
</tr>
</tbody>
</table>

Return Value

None

---

RectSetCenter

```c
void RectSetCenter(Rect *rect, Point center);
```

Purpose

Modifies the values of a `Rect` structure so that it retains its current size but is centered around the specified point.
Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>rect</th>
<th>Rect</th>
</tr>
</thead>
<tbody>
<tr>
<td>On input, specifies the size and location of a rectangle. On output, specifies a rectangle of the same size but with a different center point.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input | center  | Point |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A Point structure specifying the location of the new center point of the rectangle.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

None

RectSetFromPoints

```c
void RectSetFromPoints(Rect *rect, Point point1, Point point2);
```

Purpose

Sets the values in a Rect structure so that it defines the smallest rectangle that encloses two specified points. Each point is located on a corner of the frame of the rectangle.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>point1</th>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the location of a point.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>rect</th>
<th>Rect</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Rect structure that is set to enclose the specified points.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

None


**RectSetRight**

```c
void RectSetRight(Rect *rect, int right);
```

**Purpose**

Sets the width of a `Rect` structure so that the right edge of the rectangle it defines is at the specified location. The right edge of the rectangle is **not** enclosed by the rectangle and is equal to the left edge plus the width.

**Parameters**

<table>
<thead>
<tr>
<th>Input/Output</th>
<th><code>rect</code></th>
<th><code>Rect</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>On input, specifies the size and location of a rectangle. On output, specifies the same rectangle except with a different right edge.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th><code>right</code></th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>The x coordinate of the new right edge.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

None

---

**RectUnion**

```c
void RectUnion(Rect rect1, Rect rect2, Rect *unionRect);
```

**Purpose**

Calculates the smallest rectangle which encloses two specified rectangles.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>rect1</code></th>
<th><code>Rect</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the size and location of a rectangle.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th><code>rect1</code></th>
<th><code>Rect</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the size and location of a rectangle.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th><code>unionRect</code></th>
<th><code>Rect</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>Rect</code> structure that is set to the smallest rectangle that encloses both <code>rect1</code> and <code>rect2</code>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

None
RefreshGraph

```c
int status = RefreshGraph(int panelHandle, int controlID);
```

**Purpose**

Immediately redraws the plot area.

This action removes any plots that were deleted using `DeleteGraphPlot` in Delayed Draw mode. It also displays any plots that are plotted while `ATTR_REFRESH_GRAPH` is set to FALSE.

**Parameters**

<table>
<thead>
<tr>
<th>input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

RegisterWinMsgCallback

```c
int messageNumber = RegisterWinMsgCallback(WinMsgCallbackPtr callbackFunction, char messageIdentifier[], void *callbackData, int dataSize, int *callbackID, int deleteWhenProgramStops);
```

**Note:** This function is available only in the Windows version of LabWindows/CVI.

**Purpose**

This function registers a callback function that is called when the LabWindows/CVI application receives the specified Windows message.
### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>callbackFunction</th>
<th>WinMsgCallbackPtr</th>
<th>The name of the user function that is called whenever LabWindows/CVI receives the Windows message returned by <code>RegisterWinMsgCallback</code>. To send this message to LabWindows/CVI, call the Windows API function <code>PostMessage</code> from a DLL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageIdentifier</td>
<td>string</td>
<td>A user-defined string that allows two processes to use the same Windows message number. (See discussion below.)</td>
<td></td>
</tr>
<tr>
<td>callbackData</td>
<td>void *</td>
<td>A pointer to user-defined data passed to the event function.</td>
<td></td>
</tr>
<tr>
<td>dataSize</td>
<td>integer</td>
<td>If <code>dataSize = 0</code>, then the <code>callbackData</code> pointer is the same pointer that will be passed to <code>callbackFunction</code> when it is called. If <code>dataSize</code> is greater than 0, then the data pointed to by the <code>callbackData</code> pointer will be copied and the <code>callbackFunction</code> will be called with the <code>callbackData</code> pointer pointing to the copy of the data.</td>
<td></td>
</tr>
<tr>
<td>deleteWhenProgramStops</td>
<td>integer</td>
<td>If true, <code>UnRegisterWinMsgCallback</code> is implicitly called when the user program terminates. If false, the callback function remains installed when the user program terminates.</td>
<td></td>
</tr>
</tbody>
</table>

| Output    | callbackID       | integer | The ID that is passed to `UnRegisterWinMsgCallback` to disable the callback function. |

### Return Value

| messageNumber | integer | The message number assigned by Windows. If 0 (zero), the function failed. |

### Parameter Discussion

`callbackFunction` is a pointer to an event function that takes the form:

```c
void CVICALLBACK EventFunctionName(unsigned short wParam, unsigned lParam, void *callbackData);
```
The event function is passed the `wParam` and `lParam` Parameters that were passed to `PostMessage`. The `callbackData` that was passed to `RegisterWinMsgCallback` is also passed to the event function.

If you pass zero (0) for the `messageIdentifier`, a unique Windows message number is returned by `RegisterWinMsgCallback`. Subsequent calls to `RegisterWinMsgCallback` (or the Windows API function `RegisterWindowMessage`) does not return the same `messageNumber` until you call `UnRegisterWinMsgCallback` with the `callbackId` returned by `RegisterWinMsgCallback`.

If you pass a string for the `messageIdentifier`, any subsequent call to `RegisterWinMsgCallback` (or the Windows API function `RegisterWindowMessage`) using the same `messageIdentifier` string returns the same `messageNumber`.

To send the message registered through `RegisterWinMsgCallback` to LabWindows/CVI, pass this message number to the Windows API function `PostMessage`.

If `RegisterWinMsgCallback` fails, this value will be 0.

---

### RemovePopup

```c
int status = RemovePopup (int removePopup);
```

**Purpose**

Removes either the active pop-up panel or all pop-up panels.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>removePopup</th>
<th>integer</th>
<th>Selects whether to remove all pop-up panels or only the active pop-up panel.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = All</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Active Only.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes.                                      |
ReplaceAxisItem

```c
int status = ReplaceAxisItem(int panelHandle, int controlID, int axis,
                               int itemIndex, char itemLabel[], double itemValue);
```

Purpose

This function replaces the string/value pair at the specified index in the list of label strings for a graph or strip chart axis. These strings appear in place of the numerical labels. They appear at the location of their associated values on the graph or strip chart.

To see string labels on an X axis, you must set the ATTR_XUSE_LABEL_STRINGS attribute to TRUE. To see string labels on a Y axis, you must set the ATTR_YUSE_LABEL_STRINGS attribute to TRUE.

The original list of label strings can be created in the User Interface Editor or by calling InsertAxisItem.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>axis</td>
<td>integer</td>
<td>Specifies the axis for which to replace the string/value pair at the specified index. Valid values: VAL_XAXIS, VAL_LEFT_YAXIS, VAL_RIGHT_YAXIS (graphs only)</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index of the item to be replaced.</td>
</tr>
<tr>
<td></td>
<td>itemLabel</td>
<td>string</td>
<td>The string to replace the existing string in the item at the specified index. If you pass 0, then the existing string is not replaced. A maximum of 31 characters from the string are shown in an axis label.</td>
</tr>
<tr>
<td></td>
<td>itemValue</td>
<td>double-precision</td>
<td>The value to replace the existing value in the string/value pair at the specified index. The string appears as an axis label at the location of the value.</td>
</tr>
</tbody>
</table>
Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

InsertAxisItem, DeleteAxisItem, ClearAxisItems, GetNumAxisItems

ReplaceListItem

```c
int status = ReplaceListItem(int panelHandle, int controlID, int itemIndex,
                             char itemLabel[], ...);
```

Purpose

This function replaces the label/value pair at the specified index in a list control with a new label/value pair.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list where the item will be replaced.</td>
</tr>
<tr>
<td></td>
<td>itemLabel</td>
<td>string</td>
<td>The label to be associated with the value being replaced. Pass 0 to use the existing label.</td>
</tr>
<tr>
<td></td>
<td>itemValue</td>
<td>depends on the data type of the list control</td>
<td>The value to be associated with the label being replaced. The data type must be the same as the data type of the control.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
Parameter Discussion

The Parameter Discussion in InsertListItem tells you how to create columns and colored text in List Box controls and separator bars in Ring controls.

For picture rings, the “label” is actually an image, and you pass the pathname of the image as the itemLabel parameter. The image pathname may be a complete pathname or a simple filename. If a simple filename, the image is located in the project. If not located in the project, it is located in the directory of the project. If you pass NULL or the empty string, a placeholder for the image is created, which can be filled using ReplaceListItem or SetImageBits.

---

ReplaceTextBoxLine

```c
int status = ReplaceTextBoxLine(int panelHandle, int controlID, int lineIndex,
                                 char text[]);
```

Purpose

This function replaces the string at the specified text box line.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>lineIndex</td>
<td>integer</td>
<td>The zero-based index into the text box.</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>string</td>
<td>The string to replace the line of text at the specified lineIndex.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>
ResetTextBox

int status = ResetTextBox(int panelHandle, int controlID, char text[]);

Purpose

This function replaces all text from the specified text box with the specified string.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>string</td>
<td>The string to replace all text in the specified text box.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

ResetTimer

int status = ResetTimer(int panelHandle, int controlID);

Purpose

This function resets the interval start times for timer controls. ResetTimer resets a timer whether or not it is disabled or suspended. When a timer with an ATTR_INTERVAL of x seconds is reset, the interval is rescheduled to end x seconds from the time of the call.

You may specify an individual timer control on a panel, all timer controls on a panel, or all timer controls on all panels.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. The handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. Pass 0 to indicate all timer controls on all panels.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function. Pass 0 to indicate all timer controls on the specified panel.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

SuspendTimerCallbacks, ResumeTimerCallbacks.

ResumeTimerCallbacks

int status = ResumeTimerCallbacks(void);

Purpose

Cancels the effect of a call to SuspendTimerCallbacks.

Callbacks resume using the ongoing interval schedules, which are not affected by SuspendTimerCallbacks.

Return Value

| status | integer | Refer to Appendix A for error codes. |

See Also

SuspendTimerCallbacks, ResetTimer.
RunPopupMenu

```c
int status = RunPopupMenu (int menuBarHandle, int menuID, int panelHandle, 
    int top, int left, int pinTop, int pinLeft, int pinHeight, 
    int pinWidth);
```

**Purpose**

Displays a menu and tracks mouse and keyboard events on the menu.

In most cases, you should call this function from a User Interface panel or control callback function that has received a LEFT_CLICK, RIGHT_CLICK, or KEYPRESS event.

If the user selects an item from the menu, the following actions occur.

- If a callback function is associated with the menu item, the function is called.
- The ID of the menu item is returned.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>menuBarHandle</strong></td>
<td>integer</td>
<td>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.</td>
</tr>
<tr>
<td><strong>menuID</strong></td>
<td>integer</td>
<td>The ID for a particular menu within a menu bar. The Menu ID should be a constant name (located in the .uir header file) generated in the User Interface Editor, or a value returned by the NewMenu function.</td>
</tr>
<tr>
<td><strong>panelHandle</strong></td>
<td>integer</td>
<td>The handle for the panel over which you want the menu to appear. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td><strong>top</strong></td>
<td>integer</td>
<td>The vertical coordinate at which the upper left corner of the menu is to be placed. Must be a value from -32768 to 32767. It represents the pixel offset from the top of the panel specified by panelHandle.</td>
</tr>
<tr>
<td><strong>left</strong></td>
<td>integer</td>
<td>The horizontal coordinate at which the upper left corner of the menu is to be placed. Must be a value from -32768 to 32767. It represents the pixel offset from the left edge of the panel specified by panelHandle.</td>
</tr>
</tbody>
</table>

(continues)
Parameters (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pinTop</td>
<td>integer</td>
<td>The vertical coordinate of the upper left corner of the &quot;pin&quot; area relative to the upper left corner of the panel specified by the panelHandle parameter.</td>
</tr>
<tr>
<td>pinLeft</td>
<td>integer</td>
<td>The horizontal coordinate of the upper left corner of the &quot;pin&quot; area relative to the upper left corner of the panel specified by the panelHandle parameter.</td>
</tr>
<tr>
<td>pinHeight</td>
<td>integer</td>
<td>The height of the &quot;pin&quot; area.</td>
</tr>
<tr>
<td>pinWidth</td>
<td>integer</td>
<td>The width of the &quot;pin&quot; area.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes.                                      |

Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>ID of menu item selected.</td>
</tr>
<tr>
<td>0</td>
<td>User did not select a menu item.</td>
</tr>
</tbody>
</table>

Parameter Discussion

pinTop, pinLeft, pinHeight, and pinWidth define the "pin" area. Usually the "pin" area is the area on which the user pressed the mouse button in order to bring up the menu. When the user releases the mouse button over the "pin" area, the menu remains visible (is "pinned"). When the user releases the mouse button over any other area, the menu disappears.

If you do not want there to be a "pin" area, pass zeros for pinTop, pinLeft, pinHeight, pinWidth.

---

RunUserInterface

int status = RunUserInterface(void);

Purpose

RunUserInterface runs the User Interface and issues events to callback functions.

RunUserInterface does not return until QuitUserInterface is called from within a callback function. The value returned by RunUserInterface is the value that was passed to QuitUserInterface.
Return Value

| status  | integer | The value that was passed to the QuitUserInterface function. |

SavePanelState

```c
int status = SavePanelState (int panelHandle, char filename[], int stateIndex);
```

Purpose

Saves the state of a panel to a file. The current value of all controls on the panel are also saved. The following attributes associated with the control’s values are saved: label/value pairs; label/value pairs index; minimum, maximum, and increment values; and listbox checkmark state values.

If you want to retain plotted array data in a graph control, your original array must still be in memory when you call RecallPanelState. Alternatively, the graph could be configured to Copy Original Plot Data in the User Interface Editor or via the SetGraphAttribute function using ATTR_COPYORIGINAL_DATA.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>filename</td>
<td>string</td>
<td>The name of the file in which to save the panel state. If the file already exists, its contents will be overwritten. If the name is a simple filename (in other words, contains no directory path), then the file is saved in the directory containing the project.</td>
</tr>
<tr>
<td></td>
<td>stateIndex</td>
<td>integer</td>
<td>Assigns a unique state index to each panel state so that you can save multiple panel states to the same file.</td>
</tr>
</tbody>
</table>

Return Value

| status  | integer | Refer to Appendix A for error codes. |
SetActiveCtrl

```c
int status = SetActiveCtrl(int panelHandle, int controlID);
```

**Purpose**

This function sets the active control on the specified panel.

The active control is the control that receives keyboard events when its panel is the active panel.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
</tr>
</tbody>
</table>

- **panelHandle**: integer The specifier for a particular panel that is currently in memory. This handle will have been returned by the `LoadPanel`, `NewPanel`, or `DuplicatePanel` function.
- **controlID**: integer The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the `NewCtrl` or `DuplicateCtrl` function.

**Return Value**

| status   | integer | Refer to Appendix A for error codes. |

---

SetActiveGraphCursor

```c
int status = SetActiveGraphCursor(int panelHandle, int controlID,
int activeCursorNumber);
```

**Purpose**

Sets the active cursor on the specified graph control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>activeCursorNumber</td>
<td>integer</td>
<td>Specifies the new active cursor number. The value may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
</tr>
</tbody>
</table>

Return Value

| status  | integer | Refer to Appendix A for error codes. |

SetActivePanel

```c
int status = SetActivePanel (int panelHandle);
```

Purpose

Makes the selected panel the active panel. The active panel is the panel which receives keyboard events.

Parameters

| Input   | panelHandle   | integer | The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. |

Return Value

| status  | integer | Refer to Appendix A for error codes. |
SetAxisScalingMode

```c
int status = SetAxisScalingMode(int panelHandle, int controlID, int axis,
                                   int axisScaling, double min, double max);
```

**Purpose**

Sets the scaling mode and the range of any graph axis or the Y axis of a strip chart.

This function is not valid for the X axis of a strip chart. To set the X offset and X increment for a strip chart, use the `SetCtrlAttribute` function with the `ATTR_XAXIS_OFFSET` and `ATTR_XAXIS_GAIN` attributes.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><code>panelHandle</code></th>
<th>integer</th>
<th>The specifier for a particular panel that is contained in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>controlID</code></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td></td>
<td><code>axis</code></td>
<td>integer</td>
<td>Specifies for which axis to set the mode and range. Valid values: <code>VAL_XAXIS</code> (graphs only) <code>VAL_LEFT_YAXIS</code> (graphs and strip charts) <code>VAL_RIGHT_YAXIS</code> (graphs only)</td>
</tr>
<tr>
<td></td>
<td><code>axisScaling</code></td>
<td>integer</td>
<td>The scaling mode to be used for the axis. See table below.</td>
</tr>
<tr>
<td></td>
<td><code>min</code></td>
<td>double-precision</td>
<td>The minimum axis value when the axis is configured for manual scaling.</td>
</tr>
<tr>
<td></td>
<td><code>max</code></td>
<td>double-precision</td>
<td>The maximum axis value when the axis is configured for manual scaling.</td>
</tr>
</tbody>
</table>

**Return Value**

| `status` | integer | Refer to Appendix A for error codes. |
Parameter Discussion

axisScaling must be one of the following values.

<table>
<thead>
<tr>
<th>Valid Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_MANUAL</td>
<td>The axis is set to manual scaling, and its range is defined by min and max.</td>
</tr>
<tr>
<td>VAL_AUTOSCALE</td>
<td>The axis is set to auto scaling. min and max are not used. You cannot use auto scaling in strip charts.</td>
</tr>
<tr>
<td>VAL_LOCK</td>
<td>The axis is set to manual scaling using the current (usually auto-scaled) minimum and maximum values on the axis. You cannot use VAL_LOCK in strip charts.</td>
</tr>
</tbody>
</table>

If axisScaling is VAL_MANUAL, max must exceed min.

See Also

GetAxisScalingMode

SetAxisRange

int status = SetAxisRange (int panelHandle, int controlID, int xAxisScaling, double xminXinit, double xmaxXinc, int yAxisScaling, double ymin, double ymax);

Purpose

Sets the scaling mode and the range of the X and Y axes for a graph or strip chart control.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>xAxisScaling</td>
<td>integer</td>
<td>The scaling mode used by the X axis.</td>
</tr>
<tr>
<td></td>
<td>xminXinit</td>
<td>double-precision</td>
<td>For a graph, <em>xmin</em> specifies the minimum axis range when the X axis is configured for manual scaling. In this case, <em>xmax</em> must exceed <em>xmin</em>. For a strip chart, <em>Xinit</em> specifies the initial X axis value.</td>
</tr>
<tr>
<td></td>
<td>xmaxXinc</td>
<td>double-precision</td>
<td>For a graph, <em>xmax</em> specifies the maximum axis range when the X axis is configured for manual scaling. In this case, <em>xmax</em> must exceed <em>xmin</em>. For a strip chart, <em>Xinc</em> specifies the X axis increment for each new point.</td>
</tr>
<tr>
<td></td>
<td>yAxisScaling</td>
<td>integer</td>
<td>The scaling mode used by the Y axis.</td>
</tr>
<tr>
<td></td>
<td>ymin</td>
<td>double-precision</td>
<td>Specifies the minimum axis range when the Y axis is configured for manual scaling. In this case, <em>ymax</em> must exceed <em>ymin</em>.</td>
</tr>
<tr>
<td></td>
<td>ymax</td>
<td>double-precision</td>
<td>Specifies the maximum axis range when the Y axis is configured for manual scaling. In this case, <em>ymax</em> must exceed <em>ymin</em>.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
<td>Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>
**xAxisScaling**

<table>
<thead>
<tr>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_NO_CHANGE</td>
<td>The current X axis scaling mode remains unchanged. xmin and xmax are not used.</td>
</tr>
<tr>
<td>VAL_MANUAL</td>
<td>The X axis is set to manual scaling, and its range is defined by xmin and xmax.</td>
</tr>
<tr>
<td>VAL_AUTOSCALE</td>
<td>The X axis is set to auto scaling. xmin and xmax are not used. Auto scaling is not allowed for strip charts.</td>
</tr>
<tr>
<td>VAL_LOCK</td>
<td>The X axis is set to manual scaling using the current axis range. Lock scaling is not allowed for strip charts.</td>
</tr>
</tbody>
</table>

**yAxisScaling**

<table>
<thead>
<tr>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL_NO_CHANGE</td>
<td>The current Y axis scaling mode remains unchanged. ymin and ymax are not used.</td>
</tr>
<tr>
<td>VAL_MANUAL</td>
<td>The Y axis is set to manual scaling, and its range is defined by ymin and ymax.</td>
</tr>
<tr>
<td>VAL_AUTOSCALE</td>
<td>The Y axis is set to auto scaling. ymin and ymax are not used. Auto scaling is not allowed for strip charts.</td>
</tr>
<tr>
<td>VAL_LOCK</td>
<td>The Y axis is set to manual scaling using the current axis range. Lock scaling is not allowed for strip charts.</td>
</tr>
</tbody>
</table>

**SetCtrlAttribute**

```c
int status = SetCtrlAttribute (int panelHandle, int controlID, int controlAttribute, ...);
```

**Purpose**

Sets a control attribute for the selected panel and control.

Not all attributes are valid for each type of control, attributes may Return Values of different data types with different valid ranges. A list of attributes, their data types and valid values are provided in Table 3-9 in Chapter 3, *Programming with the User Interface Library*. 
The attributes that users cannot set are denoted as such in Chapter 3, *Programming with the User Interface Library*, Table 3-9. These attributes may be examined with the `GetCtrlAttribute` function.

**Note:** *When you set control attributes that affect the font of a control, you should modify the `ATTR_TEXT_FONT` attribute first.*

### Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the <code>.uir</code> header file) which was assigned to the control by the User Interface Editor, or the ID returned by the <code>NewCtrl</code> or <code>DuplicateCtrl</code> function.</td>
</tr>
<tr>
<td>controlAttribute</td>
<td>integer</td>
<td></td>
<td>A particular control attribute.</td>
</tr>
<tr>
<td>attributeValue</td>
<td></td>
<td></td>
<td>The value of the specified control attribute.</td>
</tr>
</tbody>
</table>

### Return Value

| status | integer | Refer to Appendix A for error codes. |

### SetCtrlBitmap

```c
int status = SetCtrlBitmap(int panelHandle, int controlID, int imageID, int bitmapID);
```

**Purpose**

Sets the image of a control from a bitmap object. Can be used to replace an existing image on a control or to create a new image on a control.

The following control types can contain images:

- picture controls
- picture rings
- picture buttons
- graph controls

For picture controls, this function can be used as an alternative to `DisplayImageFile`. 
For picture buttons, this function can be used as an alternative to SetCtrlAttribute with the attribute set to ATTR_IMAGE_FILE.

For picture rings, this function can be used as an alternative to ReplaceListItem. (To add a new entry, first call InsertListItem with a NULL value, and then call SetCtrlBitmap.)

For graphs, you must first call PlotBitmap with a NULL filename. Then call SetCtrlBitmap.

If you want to delete an image, call SetCtrlBitmap with 0 as the value for the bitmap ID.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is contained in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control in the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>imageID</td>
<td>integer</td>
<td>For picture rings, the zero-based index of an image in the ring. For graphs, this argument is the plotHandle returned from PlotBitmap. For picture controls and picture buttons, this argument is ignored.</td>
</tr>
<tr>
<td>bitmapID</td>
<td>integer</td>
<td>The ID of the bitmap object containing the new image. The ID must have been obtained from NewBitmap, GetBitmapFromFile, GetCtrlBitmap, ClipboardGetBitmap, GetCtrlDisplayBitmap, GetPanelDisplayBitmap.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
</tr>
</tbody>
</table>

Refer to Appendix A for error codes.

**See Also**


**SetCtrlIndex**

```c
int status = SetCtrlIndex(int panelHandle, int controlID, int itemIndex);
```
Purpose

This function sets the current index of the specified list control.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list control.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

SetCtrlVal

int status = SetCtrlVal (int panelHandle, int controlID, ...);

Purpose

Sets the value of a control to the specified value.

When called on a list control (a list box or ring), SetCtrlVal sets the current list item to the first item whose associated value is value. To set the current list item via zero-based index, use the SetCtrlIndex function.

When called on a text box, SetCtrlVal appends value to the contents of the text box. Use ResetTextBox to replace the contents of the text box with value.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
</table>
### SetCursorAttribute

```c
int status = SetCursorAttribute(int panelHandle, int controlID, int cursorNumber,
                                int cursorAttribute, int attributeValue);
```

**Purpose**

Sets one of the following graph cursor attributes: mode, point style, cross hair style, color, or y-axis.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>parameter</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
<td></td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
<td></td>
</tr>
<tr>
<td>cursorNumber</td>
<td>integer</td>
<td>The cursor number may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
<td></td>
</tr>
<tr>
<td>cursorAttribute</td>
<td>integer</td>
<td>Selects a particular graph cursor attribute. Valid Attributes: ATTR_CURSOR_MODE, ATTR_CURSOR_POINT_STYLE, ATTR_CROSS_HAIR_STYLE, ATTR_CURSOR_COLOR, ATTR_CURSOR_YAXIS</td>
<td></td>
</tr>
<tr>
<td>attributeValue</td>
<td>integer</td>
<td>The new value of the attribute. See Table 3-20 for a complete listing of cursor attribute values.</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

SetFontPopupDefaults

```c
int status = SetFontPopupDefaults(char typefaceName[], int bold, int underline, int strikeOut, int italic, int justification, int textColor, int fontSize);
```

Purpose

This function specifies the settings to apply to the FontSelectPopup when the user presses the Default button.

This function applies to all attributes that affect the sample text display, even attributes for which the controls have been hidden.
The default values specified by this function apply only to the next call to `FontSelectPopup`. Thus, this function should always be called before each call to `FontSelectPopup`. If you do not call it before `FontSelectPopup`, the parameters have the following default values:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typefaceName</code></td>
<td><code>VAL_DIALOG_FONT</code></td>
</tr>
<tr>
<td><code>bold</code></td>
<td>0</td>
</tr>
<tr>
<td><code>underline</code></td>
<td>0</td>
</tr>
<tr>
<td><code>strikeOut</code></td>
<td>0</td>
</tr>
<tr>
<td><code>italic</code></td>
<td>0</td>
</tr>
<tr>
<td><code>justification</code></td>
<td><code>VAL_LEFT_JUSTIFIED</code></td>
</tr>
<tr>
<td><code>textColor</code></td>
<td><code>VAL_BLACK</code></td>
</tr>
<tr>
<td><code>fontSize</code></td>
<td>12</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><code>typefaceName</code></th>
<th>string</th>
<th>The default typeface name (for example, &quot;Courier&quot;) for use by <code>FontSelectPopup</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bold</code></td>
<td>integer</td>
<td></td>
<td>The default bold setting for use by <code>FontSelectPopup</code>.</td>
</tr>
<tr>
<td><code>underline</code></td>
<td>integer</td>
<td></td>
<td>The default underline setting for use by <code>FontSelectPopup</code>.</td>
</tr>
<tr>
<td><code>strikeOut</code></td>
<td>integer</td>
<td></td>
<td>The default strike out setting for use by <code>FontSelectPopup</code>.</td>
</tr>
<tr>
<td><code>italic</code></td>
<td>integer</td>
<td></td>
<td>The default italic setting for use by <code>FontSelectPopup</code>.</td>
</tr>
<tr>
<td><code>justification</code></td>
<td>integer</td>
<td></td>
<td>The default justification setting for use by <code>FontSelectPopup</code>.</td>
</tr>
<tr>
<td><code>textColor</code></td>
<td>integer</td>
<td></td>
<td>The default text color setting for use by <code>FontSelectPopup</code>.</td>
</tr>
<tr>
<td><code>fontSize</code></td>
<td>integer</td>
<td></td>
<td>The default font size for use by <code>FontSelectPopup</code>.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

…
See Also
FontSelectPopup.

SetGraphCursor

```c
int status = SetGraphCursor (int panelHandle, int controlID, int cursorNumber, double x, double y);
```

Purpose

Sets the position of the specified graph cursor.

The position is relative to the current range of the X and Y axes.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
<td></td>
</tr>
<tr>
<td>cursorNumber</td>
<td>integer</td>
<td>The cursor number may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>double-precision</td>
<td>Specifies the X coordinate of the new cursor position.</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>double-precision</td>
<td>Specifies the Y coordinate of the new cursor position.</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
SetGraphCursorIndex

```c
int status = SetGraphCursorIndex (int panelHandle, int controlID, int cursorNumber,
         int plotHandle, int arrayIndex);
```

Purpose

Attaches a cursor to particular data point defined by a plot handle and array index.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>controlID</strong></td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td><strong>cursorNumber</strong></td>
<td>integer</td>
<td>The cursor number may range from 1 to the number of defined cursors for the specified graph. The number of defined cursors is established when you edit the graph in the User Interface Editor or through SetCtrlAttribute.</td>
</tr>
<tr>
<td></td>
<td><strong>plotHandle</strong></td>
<td>integer</td>
<td>Specifies the handle of the plot on which to attach the cursor. Cursors cannot be attached to plots generated by, PlotText, PlotOval, PlotRectangle, PlotArc, PlotLine, PlotBitmap, PlotIntensity</td>
</tr>
<tr>
<td></td>
<td><strong>arrayIndex</strong></td>
<td>integer</td>
<td>Specifies the array index of the data point on which to attach the cursor.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |
SetIdleEventRate

int status = SetIdleEventRate (int interval);

Purpose

This function sets the interval between idle events.

Idle events can be processed by the main callback function if enabled through InstallMainCallback.

The InstallMainCallback function takes the name of an event function (type MainCallbackPtr), a pointer to callback data of any type, and a Boolean indicating whether or not to get idle events.

The getIdleEvents feature allows program execution to proceed when it would normally be suspended. For example, the program would normally be suspended in GetUserEvent or RunUserInterface when the user holds the mouse button down on a control or pull-down menu.

Note: Idle events are not generated while a top level panel is being moved or sized.

Parameters

| Input | interval | integer | Specifies the wait interval between generated idle events (in milliseconds). A value of zero will cause idle events to occur at the fastest possible rate. For non-zero values, the resolution of idle events is the resolution of the system timer (1 milliseconds on Windows) and an operating system dependent latency. |

Return Value

| status | integer | Refer to Appendix A for error codes. |
SetImageBits

```c
int status = SetImageBits(int panelHandle, int controlID, int imageID,
                        int rowBytes, int depth, int width, int height,
                        int colorTable[], unsigned char bitmap[],
                        unsigned char mask[]);
```

**Purpose**

Sets the bit values that define an image. Can be used to replace an existing image on a control or to create a new image on a control.

The following control types can contain images: picture controls, picture rings, picture buttons, graph controls.

For a picture control, this function can be used as an alternative to `DisplayImageFile`.

For a picture buttons, this function can be used as an alternative to `SetCtrlAttribute` with the attribute `ATTR_IMAGE_FILE`.

For picture rings, this function can be used as an alternative to `ReplaceListItem`. (To add a new entry, first call `InsertListItem` with a NULL value, and then call `SetImageBits`.)

For graphs, you must first call `PlotBitmap` with a NULL filename. Then call `SetImageBits`.

If you want to delete an image, call `SetImageBits` with -1 as the value for the `width` or the `height` parameter.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>panelHandle</td>
<td>integer</td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>imageID</td>
<td>integer</td>
<td>For a picture ring, the zero-based index of an image in the ring. For a graph, the plotHandle, returned from PlotBitmap. For picture controls and buttons, this is ignored.</td>
</tr>
<tr>
<td></td>
<td>rowBytes</td>
<td>integer</td>
<td>The number of bytes on each scan line of the image.</td>
</tr>
<tr>
<td></td>
<td>depth</td>
<td>integer</td>
<td>The number of bits per pixel.</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>integer</td>
<td>The width of the image, in pixels.</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>integer</td>
<td>The height of the image, in pixels.</td>
</tr>
<tr>
<td></td>
<td>colorTable</td>
<td>integer array</td>
<td>An array of RGB color values.</td>
</tr>
<tr>
<td></td>
<td>bitmap</td>
<td>unsigned char array</td>
<td>An array of bits that determine the colors to be displayed on each pixel in the image.</td>
</tr>
<tr>
<td></td>
<td>mask</td>
<td>unsigned char array</td>
<td>An array containing one bit per pixel in the image. Each bit specifies whether the pixel is actually drawn.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
<td>Refer to Appendix A for error codes.</td>
</tr>
</tbody>
</table>

Parameter Discussion

If either the width or height parameter is -1 and there is an existing image, it is deleted.

Depending on the depth and width, the number of bits per scan line in the bitmap array may not be an even multiple of 8. If not, then the extra bits needed to get to the next byte boundary is considered "padding". If you specify rowBytes as a positive number, then the bits for each scan line must start on a byte boundary, and so padding may be required. In fact, you can specify a value for rowBytes that is larger than the minimum number of bytes actually needed. The extra
bytes are also considered padding. If you pass -1, there is no padding at all. The bits for each
scan line immediately follow the bits for the previous scan line.

The valid values for **pixelDepth** are 1, 4, 8, 24, and 32.

If the **pixelDepth** is 8 or less, the number of entries in the **colorTable** array must equal 2 raised
to the power of the **pixelDepth** parameter. The **bits** array contain indices into the **colorTable**
array. If the **pixelDepth** is greater than 8, the **colorTable** parameter is not used. Instead the **bits**
array contains actual RGB color values, rather than indices into the **colorTable** array.

For a **pixelDepth** of 24, each pixel is represented by a 3-byte RGB value of the form
0xRRGGBB, where RR, GG, and BB represent the red, green and blue intensity of the color. The
RR byte should always be at the lowest memory address of the three bytes.

If the **pixelDepth** is 32, each pixel in the **bits** array is represented by a 32-bit RGB value of the
form 0x00RRGGBB, where RR, GG, and BB represent the red, green and blue intensity of the
color. The 32-bit value is treated as a native 32-bit integer value for the platform. The most
significant byte is always ignored. The BB byte should always be in the least significant byte.
On a little-endian platform (for example, Intel processors), BB would be at the lowest memory
address. On a big-endian platform (for example, Motorola processors), BB would be at the
highest address. Note that this differs from the format of the bits array when the **pixelDepth** is
24.

In the **mask** array, a bit value of 1 indicates that the pixel is drawn. 0 indicates that the pixel is
not drawn. Exception: If an image has a **pixelDepth** of 1, pixels with a **bits** value of 1
(foreground pixels) are always drawn and the **mask** affects only the pixels with a bitmap of 0
(background pixels). Each row of the mask must be padded to the nearest even-byte boundary.
For example, if the width of the image is 21 pixels, then there must be 32 bits (in other words, 4
bytes) of data in each row of the mask.

A mask is useful for achieving transparency.

You may pass NULL if you do not need a mask.

**See Also**

GetImageInfo, AllocImageBits, GetImageBits.
SetInputMode

```c
int status = SetInputMode(int panelorMenuBarHandle, int controlorMenuItemID, int inputMode);
```

**Purpose**

Controls whether user input is recognized.

When an object’s input mode is disabled, it appears dimmed on the screen. Recognition of user input can be controlled at the panel, control, menu bar, menu, or menu item level.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelorMenuBarHandle</th>
<th>integer</th>
<th>The handle of the panel or menu bar on which you wish to set the recognition of user input. If -1 is entered, all panels and all menu bars are affected.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlorMenuItemID</td>
<td>integer</td>
<td>The ID of the control or menu item on which you wish to set the recognition of user input. The ID is the defined constant that was assigned to the control or menu item in the User Interface Editor or the ID returned by NewCtrl or NewMenuItem. If -1, all controls on the panel and/or all items in the menu bar are affected.</td>
</tr>
<tr>
<td></td>
<td>inputMode</td>
<td>integer</td>
<td>Specifies whether user input is enabled or disabled. 0 = disabled. 1 = enabled.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>
SetListItemImage

```
int status = SetListItemImage(int panelHandle, int controlID, int itemIndex, int image);
```

**Purpose**

This function puts a predefined image in a list control on the same line as the specified item index.

This function is not valid for picture ring controls. For picture rings, see `setImageBits`.

**Valid Images:**

| no folder          | VAL_NO_IMAGE  |
|                   |               |
| folder            | VAL_FOLDER    |
| open folder       | VAL_OPEN_FOLDER |
| current folder    | VAL_CURRENT_FOLDER |

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>itemIndex</td>
<td>integer</td>
<td>The zero-based index into the list.</td>
</tr>
<tr>
<td></td>
<td>image</td>
<td>integer</td>
<td>Selects an image for display in the specified list item.</td>
</tr>
</tbody>
</table>

**Return Value**

| status         | integer | Refer to Appendix A for error codes. |
SetMenuBarAttribute

```c
int status = SetMenuBarAttribute (int menuBarHandle, int menuorMenuItemID,
                                    int menuBarAttribute, ...);
```

**Purpose**

This function sets the value of the specified menu bar attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>menuBarHandle</th>
<th>integer</th>
<th>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>menuorMenuItemID</td>
<td>integer</td>
<td>The menu or menu item ID assigned in the User Interface Editor or returned by the NewMenu or NewMenuItem. If the attribute corresponds to the entire menu bar, pass 0 as this parameter.</td>
</tr>
<tr>
<td></td>
<td>menuBarAttribute</td>
<td>integer</td>
<td>See Table 3-6 in Chapter 3 for a complete listing of menu bar attributes.</td>
</tr>
<tr>
<td></td>
<td>attributeValue</td>
<td>depends on the attribute</td>
<td>The value of the selected menu bar attribute. See Table 3-6 in Chapter 3 for a complete listing of menu bar attribute values.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

SetMouseCursor

```c
int status = SetMouseCursor (int mouseCursorStyle);
```

**Purpose**

Sets the mouse cursor to the specified style.

This function sets the mouse cursor for all existing panels and any panels created by NewPanel or DuplicatePanel or loaded by LoadPanel.
Parameters

| Input | mouseCursorStyle | integer | Specifies the mouse cursor style. See Table 3-4 in Chapter 3 for a list of valid mouse cursor styles. |

Return Value

| status | integer | Refer to Appendix A for error codes. |

SetPanelAttribute

```
int status = SetPanelAttribute(int panelHandle, int panelAttribute, ...);
```

Purpose

This function sets a particular panel attribute.

Parameters

| Input | panelHandle | integer | The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function. |
|       |            |        | |
|       | panelAttribute | integer | A particular panel attribute. See Table 3-2 in Chapter 3 for a complete listing of panel attributes. |
|       | attributeValue | depends on the attribute | The value of the specified panel attribute. See Table 3-2 in Chapter 3 for a complete listing of panel attribute values. |

Return Value

| status | integer | Refer to Appendix A for error codes. |
### SetPanelMenuBar

```c
int status = SetPanelMenuBar(int panelHandle, int menuBarHandle);
```

**Purpose**

This function assigns a menu bar to the specified panel.

A panel can have only one menu bar at a time, but multiple panels can share the same menu bar.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>menuBarHandle</td>
<td>integer</td>
<td>The specifier for a particular menu bar that is currently in memory. This handle will have been returned by LoadMenuBar or NewMenuBar.</td>
</tr>
</tbody>
</table>

**Return Value**

| status  | integer | Refer to Appendix A for error codes. |

### SetPanelPos

```c
int status = SetPanelPos(int panelHandle, int panelTop, int panelLeft);
```

**Purpose**

Sets the position of upper left corner (directly below the title bar) of the panel. The size of the panel remains constant.

The size of the panel may be changed with the SetPanelAttribute function.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>panelTop</td>
<td>integer</td>
<td>The vertical coordinate at which the upper left corner of the panel (directly below the title bar) is placed.</td>
</tr>
<tr>
<td>Input</td>
<td>panelLeft</td>
<td>integer</td>
<td>The horizontal coordinate at which the upper left corner of the panel (directly below the title bar) is placed.</td>
</tr>
</tbody>
</table>
Return Value

| status    | integer | Refer to Appendix A for error codes. |

Parameter Discussion

The `panelTop` and `panelLeft` coordinates must be integer values from -32768 to 32767, or `VAL_AUTO_CENTER` to center the panel. For a top-level panel, (0,0) is the upper-left corner of the screen. For a child panel, (0,0) is the upper-left corner of the parent panel (directly below the title bar) before the parent panel is scrolled.

---

**SetPlotAttribute**

```c
int status = SetPlotAttribute (int panelHandle, int controlID, int plotHandle,
                               int plotAttribute, ...);
```

**Purpose**

Sets the value of a graph plot attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>panelHandle</th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>controlID</td>
<td>integer</td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td></td>
<td>plotHandle</td>
<td>integer</td>
<td>The handle for a particular plot in the graph. It will have been returned by one of the graph plotting functions.</td>
</tr>
<tr>
<td></td>
<td>plotAttribute</td>
<td>integer</td>
<td>Selects a particular graph plot attribute. See Table 3-20 in Chapter 3 for a complete listing of plot attributes.</td>
</tr>
<tr>
<td></td>
<td>attributeValue</td>
<td>depends</td>
<td>The value of the specified plot attribute. See Table 3-20 in Chapter 3 type of for a complete listing of plot attributes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on the attribute</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

| status   | integer | Refer to Appendix A for error codes. |

---

SetPrintAttribute

```c
int status = SetPrintAttribute(int printAttribute, int attributeValue);
```

Purpose

Sets the value of the selected print attribute.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>printAttribute</th>
<th>integer</th>
<th>A particular print attribute. See Table 3-27 in Chapter 3 for a complete listing of print attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>attributeValue</td>
<td>integer</td>
<td>The new value of the print attribute. See Table 3-27 in Chapter 3 for a complete listing of print attribute values.</td>
</tr>
</tbody>
</table>

Return Value

| status   | integer | Refer to Appendix A for error codes. |

---

SetSleepPolicy

```c
int status = SetSleepPolicy(int sleepPolicy);
```

Purpose

Each time the User Interface Library checks for an event from the operating system, it may put your program in the background ("to sleep") for a specified period of time. The benefit is that other applications are given more processor time. The disadvantage is that your program may run slower.

This function sets the degree your program "sleeps" when checking for events. The setting that is optimal for your program depends on the operating system you are using and the other applications you are running. If you think you may need to make an adjustment, try the different settings and observe the resulting behavior.
Parameters

| Input   | sleepPolicy | integer | The degree to which your program periodically sleeps each time the User Interface Library checks for events from the operating system. |

Return Value

| status  | integer | Refer to Appendix A for error codes. |

Parameter Discussion

Sleep policy may be one of the following:

| VAL_SLEEP_NONE | 1 | Never be put to sleep. The default value. |
| VAL_SLEEP_SOME | 2 | Be put to sleep a small period. |
| VAL_SLEEP_MORE | 3 | Be put to sleep for a longer period. |

SetSystemAttribute

```c
int status = SetSystemAttribute (int systemAttribute, ...);
```

Purpose

Sets the value of a particular system attribute.

Parameters

| Input   | systemAttribute | integer | ATTR_ALLOW_UNSAFE_TIMER_EVENTS
|        | attributeValue   | depends on the attribute | ATTR_REPORT_LOAD_FAILURE
|        |                 |                        | ATTR_ALLOW_MISSING_CALLBACKS |
|        |                 |                        | The value of the specified attribute. See Table 3-26 for values associated with this attribute. |

Return Value

| status  | integer | Refer to Appendix A for error codes. |
SetSystemPopupsAttribute

```c
int status = SetSystemPopupsAttribute (int popupAttribute, ...);
```

**Purpose**

Sets the value of a particular system pop-up attribute. All subsequent system pop-ups inherit the new attribute value.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>popupAttribute</td>
<td>integer ATTR_MOVABLE or ATTR_SYSTEM_MENU_VISIBLE.</td>
<td>The value of the specified attribute. See Table 3-2 in Chapter 3 for valid values associated with these attributes.</td>
</tr>
<tr>
<td>attributeValue</td>
<td>depends on the attribute</td>
<td></td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

SetTraceAttribute

```c
int status = SetTraceAttribute (int panelHandle, int controlID, int traceNumber, int traceAttribute, int attributeValue);
```

**Purpose**

Sets the value of a particular strip chart trace attribute.
Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelHandle</td>
<td>integer</td>
<td></td>
<td>The specifier for a particular panel that is currently in memory. This handle will have been returned by the LoadPanel, NewPanel, or DuplicatePanel function.</td>
</tr>
<tr>
<td>controlID</td>
<td>integer</td>
<td></td>
<td>The defined constant (located in the .uir header file) which was assigned to the control by the User Interface Editor, or the ID returned by the NewCtrl or DuplicateCtrl function.</td>
</tr>
<tr>
<td>traceNumber</td>
<td>integer</td>
<td></td>
<td>The strip chart trace number may be from 1 to the number of defined strip chart traces. The number of defined strip chart traces is established through editing the strip chart in the User Interface Editor or calling SetCtrlAttribute.</td>
</tr>
<tr>
<td>traceAttribute</td>
<td>integer</td>
<td></td>
<td>Selects a particular strip chart trace attribute. See Table 3-20 in Chapter 3 for a complete listing of trace attributes.</td>
</tr>
<tr>
<td>attributeValue</td>
<td>integer</td>
<td></td>
<td>The current value of the trace attribute.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

SetWaitCursor

```c
int status = SetWaitCursor (int waitCursorState);
```

Purpose

Specifies the state of the wait cursor. If the wait cursor is activated, all other cursor styles are overridden to display the wait cursor.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>waitCursorState</td>
<td>integer</td>
<td></td>
<td>Specifies the state of the wait cursor. When the wait cursor is active, all other cursor styles are overridden until cursor is deactivated. 1 = wait cursor active, 0 = wait cursor inactive.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

SuspendTimerCallbacks

```c
int status = SuspendTimerCallbacks(void)
```

Purpose

This function stops all timer callbacks until a call is made to ResumeTimerCallbacks. The function does not alter the ongoing schedule of timer intervals. It only inhibits the calling of callback functions.

Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>

See Also

ResumeTimerCallbacks, ResetTimer

UnRegisterWinMsgCallback

```c
int status = UnRegisterWinMsgCallback(int callbackID);
```

Purpose

**Note:** This function is available only on the Windows version of LabWindows/CVI.

This function registers a callback function that will be called when the LabWindows/CVI application receives the specified Windows message.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>callbackID</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The ID that was returned in the callbackID parameter of RegisterWinMsgCallback.</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>status</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = Successfully unregistered the callback function</td>
<td></td>
</tr>
<tr>
<td>0 = Failed to unregister the callback function.</td>
<td></td>
</tr>
</tbody>
</table>
ValidatePanel

```c
int status = ValidatePanel(int panelHandle, int *valid);
```

**Purpose**

Verifies that the value of each numeric control in notify mode is within the range specified in the User Interface Editor or set by the `SetCtrlAttribute` function using `ATTR_MAX_VALUE` and `ATTR_MIN_VALUE`.

If a control is out of range, a red outline appears around it prompting the user to enter a new value.

When the control has the input focus, a pop-up appears indicating the control’s range and default value.

**Note:** *This function is automatically called when a user attempts to generate a commit event on a validate control.*

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th><strong>panelHandle</strong></th>
<th>integer</th>
<th>The specifier for a particular panel that is currently in memory. This handle will have been returned by the <code>LoadPanel</code>, <code>NewPanel</code>, or <code>DuplicatePanel</code> function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td><strong>valid</strong></td>
<td>integer</td>
<td>A Boolean value indicating whether or not the panel is valid. If valid, then each control is valid. If invalid, then at least one and possibly more controls are invalid. 0 = one or more controls are out of range 1 = all controls are in range.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th><strong>status</strong></th>
<th>integer</th>
<th>Refer to Appendix A for error codes.</th>
</tr>
</thead>
</table>
WaveformGraphPopup

```c
int status = WaveformGraphPopup(char title[], void *yArray, int numberOfPoints,
                                  int yDataType, double yGain, double yOffset,
                                  double initialX, double xIncrement);
```

**Purpose**

Plots a waveform on a dialog box.

The values in `yArray` are scaled according to `yGain` and `yOffset`. The `Xaxis` timebase is scaled according to `initialX` and `xIncrement`. Each point in the plot is computed as follows:

\[
x_i = (i \times XInc) + InitX
\]

\[
y_i = (wfm \times YGain) + Yoff
\]

where `i` is the index of the point in the waveform array.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>The title to be displayed on the dialog box.</td>
</tr>
<tr>
<td>yArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by <code>yDataType</code>.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in <code>xArray</code> is greater than the <code>numberOfPoints</code>.</td>
</tr>
<tr>
<td>yDataType</td>
<td>integer</td>
<td>Specifies the data type of the <code>yArray</code>. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td>yGain</td>
<td>double-precision</td>
<td>Specifies the gain to be applied to the waveform <code>yArray</code> data.</td>
</tr>
<tr>
<td>yOffset</td>
<td>double-precision</td>
<td>Specifies a constant offset to be added to the waveform <code>yArray</code> data.</td>
</tr>
<tr>
<td>initialX</td>
<td>double-precision</td>
<td>Specifies the initial value for the X axis.</td>
</tr>
<tr>
<td>xIncrement</td>
<td>double-precision</td>
<td>Specifies the increment along the X axis for each new point.</td>
</tr>
</tbody>
</table>

**Return Value**

| status | integer | Refer to Appendix A for error codes. |

© National Instruments Corporation 4-251 LabWindows/CVI User Interface Reference
XGraphPopup

```c
int status = XGraphPopup(char *title[], void *xArray, int numberOfPoints,
                           int xDataType);
```

**Purpose**

Plots an array of X values against its indices along the Y axis in a dialog box.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td></td>
<td>The title to be displayed in the dialog box.</td>
</tr>
<tr>
<td>xArray</td>
<td>void *</td>
<td></td>
<td>The array that contains the values to be plotted along the X axis. The data type must be of the type specified by xDataType.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td></td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in xArray is greater than the numberOfPoints.</td>
</tr>
<tr>
<td>xDataType</td>
<td>integer</td>
<td></td>
<td>Specifies the data type of the xArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
</tbody>
</table>

**Return Value**

| status   | integer | Refer to Appendix A for error codes. |

---

XYGraphPopup

```c
int status = XYGraphPopup(char *title[], void *xArray, void *yArray,
                           int numberOfPoints, int xDataType, int yDataType);
```

**Purpose**

Plots an array of Y values against an array of X values in a dialog box.

**Parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td></td>
<td>The title to be displayed in the dialog box.</td>
</tr>
<tr>
<td>xArray</td>
<td>void *</td>
<td></td>
<td>The array that contains the values to be plotted along the X axis. The data type must be of the type specified by xDataType.</td>
</tr>
</tbody>
</table>

(continues)
Parameters (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yArray</td>
<td>void *</td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by yDataType.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in xArray is greater than the numberOfPoints.</td>
</tr>
<tr>
<td>xDataType</td>
<td>integer</td>
<td>Specifies the data type of the xArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
<tr>
<td>yDataType</td>
<td>integer</td>
<td>Specifies the data type of the yArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

YGraphPopup

```c
int status = YGraphPopup(char *title[], void *yArray, int numberOfPoints, int yDataType);
```

Purpose

Plots an array of Y values against its indices along the X axis in a dialog box.

Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>title</th>
<th>string</th>
<th>The title to be displayed on the dialog box.</th>
</tr>
</thead>
<tbody>
<tr>
<td>yArray</td>
<td>void *</td>
<td></td>
<td>The array that contains the values to be plotted along the Y axis. The data type must be of the type specified by yDataType.</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>integer</td>
<td></td>
<td>The number of points to plot. This value controls the number of points to plot even if the number of elements in xArray is greater than the numberOfPoints.</td>
</tr>
<tr>
<td>yDataType</td>
<td>integer</td>
<td></td>
<td>Specifies the data type of the yArray. See Table 3-11 in Chapter 3 for a list of data types.</td>
</tr>
</tbody>
</table>

Return Value

| status | integer | Refer to Appendix A for error codes. |

© National Instruments Corporation 4-253 LabWindows/CVI User Interface Reference
Chapter 5
LabWindows/CVI Sample Programs

When you installed LabWindows/CVI, you had the option to install a collection of sample programs that demonstrate various concepts of the User Interface Library. If you selected this installation option, the programs were copied to the samples subdirectory. These sample programs supplement the information presented in this manual. For more information on User Interface and other example programs, see the samples.doc text file in LabWindows/CVI’s main directory.

Example Program Files

Table 5-1 is a quick reference guide outlining the sample programs and the concepts that they illustrate.

Table 5-1. Sample Program Files

<table>
<thead>
<tr>
<th>Item Number</th>
<th>Project File Name</th>
<th>Program Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>io.prj</td>
<td>Standard I/O</td>
</tr>
<tr>
<td>2.</td>
<td>callback.prj</td>
<td>Introduction to Callback Functions</td>
</tr>
<tr>
<td>3.</td>
<td>events.prj</td>
<td>User Interface Events</td>
</tr>
<tr>
<td>4.</td>
<td>menus.prj</td>
<td>Menus Controlled by Callback Functions</td>
</tr>
<tr>
<td>5.</td>
<td>graphs.prj</td>
<td>Graphs</td>
</tr>
<tr>
<td>6.</td>
<td>chart prj</td>
<td>Strip Charts</td>
</tr>
<tr>
<td>7.</td>
<td>cursors.prj</td>
<td>Graph Cursor</td>
</tr>
<tr>
<td>8.</td>
<td>popups.prj</td>
<td>Pop-Up Panels</td>
</tr>
<tr>
<td>9.</td>
<td>listbox.prj</td>
<td>Selection Lists</td>
</tr>
<tr>
<td>10.</td>
<td>panels.prj</td>
<td>Child Windows</td>
</tr>
<tr>
<td>11.</td>
<td>timerctl.prj</td>
<td>Timer Controls</td>
</tr>
<tr>
<td>12.</td>
<td>textbox.prj</td>
<td>Text Boxes</td>
</tr>
</tbody>
</table>

(continues)
Table 5-1. Sample Program Files (Continued)

<table>
<thead>
<tr>
<th>Item Number</th>
<th>Project File Name</th>
<th>Program Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>picture.prj</td>
<td>Using Picture Controls</td>
</tr>
<tr>
<td>14.</td>
<td>build.prj</td>
<td>Building a User Interface Programmatically</td>
</tr>
<tr>
<td>15.</td>
<td>getusrev.prj</td>
<td>Programming with Event Loops</td>
</tr>
<tr>
<td>16.</td>
<td>keyfiltr.prj</td>
<td>Handling Keyboard Input</td>
</tr>
<tr>
<td>17.</td>
<td>moustate.prj</td>
<td>Getting the Mouse State</td>
</tr>
<tr>
<td>18.</td>
<td>listdelx.prj</td>
<td>Colors in List Boxes</td>
</tr>
<tr>
<td>19.</td>
<td>2yaxis.prj</td>
<td>Two Y Axes on a Graph</td>
</tr>
<tr>
<td>20.</td>
<td>intgraph.prj</td>
<td>Intensity Plots</td>
</tr>
<tr>
<td>21.</td>
<td>autostrp.prj</td>
<td>Autoscaling the Y-Axis on a Strip Chart</td>
</tr>
<tr>
<td>22.</td>
<td>canvas.prj</td>
<td>Canvas Controls</td>
</tr>
<tr>
<td>23.</td>
<td>canvsbmk.prj</td>
<td>Canvas Benchmark</td>
</tr>
<tr>
<td>24.</td>
<td>drawpad.prj</td>
<td>Using Canvas as Drawing Pad</td>
</tr>
<tr>
<td>25.</td>
<td>piedemo.prj</td>
<td>Pie Chart</td>
</tr>
<tr>
<td>26.</td>
<td>imagedit.prj</td>
<td>Changing Image Colors</td>
</tr>
<tr>
<td>27.</td>
<td>clipbord.prj</td>
<td>Using System Clipboard</td>
</tr>
</tbody>
</table>

Using the Sample Programs

Each of these sample programs are meant to illustrate a particular feature or concept of the LabWindows/CVI User Interface Library. The source code is meant to be simple and easy to read, so that you can use these sample programs to learn how to use particular controls and event processing concepts, and act as a guideline for developing your own applications.

To examine or run a sample program, load the project file corresponding to the desired sample into the Project Window. Remember, all sample programs are in the samples\userint subdirectory. We recommend that you review the sample programs in the order presented in Table 5-1. The first few samples highlight fundamental concepts for building programs in LabWindows/CVI, while the later samples illustrate more specialized features of the User Interface Library.
1. io.prj—Standard I/O

This sample program demonstrates how to display and retrieve information from users through the Standard Input/Output window using the ANSI C `stdio` library functions.

2. callback.prj—Introduction to Callback Functions

This sample program demonstrates how to use callback functions to respond to events generated on a LabWindows/CVI user interface.

3. events.prj—User Interface Events

This sample program demonstrates how you can respond to multiple events generated from a single control on the user interface. For example, this sample demonstrates how your programs can recognize the difference between a left mouse click and right mouse click occurring on a panel. This sample also shows how to gain access to supplementary event information, such as the X and Y-coordinates of the mouse cursor when the click occurred.

4. menus.prj—Menus Controlled by Callback Functions

This sample illustrates some of the ways that you can build menus in the menu editor in the User Interface Editor, and how to respond to these menu selections through a menu callback function.

5. graphs.prj—Graphs

This sample demonstrates the various graphing routines available in the User Interface Library.

6. chart.prj—Strip Charts

This sample demonstrates how to display multiple traces on a strip chart control.

7. cursors.prj—Graph Cursors

This sample program demonstrates how to use cursors for zooming operations on a graph control, and how to use snap-to-point cursor to get X- and Y-coordinate information from the graph.

8. popups.prj—Pop-Up Panels

This sample program demonstrates the various pop-up panel controls available in the LabWindows/CVI User Interface Library.
9. listbox.prj—Selection Lists

This sample program demonstrates how to select, add, and delete items from a list box control programmatically.

10. panels.prj—Child Windows

This sample program demonstrates how you can define and display child windows in LabWindows/CVI.

11. timerctl.prj—Timer Controls

This sample program demonstrates how you can use a timer control to perform a particular action continuously without being affected by other user interface operations. For example, you can plot data to a strip chart continuously without hesitation, even when a menu bar is pulled down, or a panel is popped up on top of the strip chart.

12. textbox.prj—Text Boxes

This sample program demonstrates how you can use a text box control to display help and status information. You see the fundamental actions you can perform on a text box such as adding a new line of text, inserting a line of text, and deleting text.

13. picture.prj—Using Picture Controls

This sample program illustrates how you can use picture controls to enhance a user interface. The program contains two PCX images imported into two picture controls. One image sits on top of and covers the other. A left click event on the picture control causes a callback function to bring the hidden image to the foreground, covering the other image. You could follow the example of this sample program to create a toggle button that has different images for the on and off states.

14. build.prj—Building a User Interface Programmatically

This sample program demonstrates how to build a user interface from your program using the NewPanel and NewCtrl functions, and how to install callback functions manually with the InstallCtrlCallback function. It also illustrates how to change customize controls programmatically using the SetCtrlAttribute function.
15. getusrev.prj—Programming with Event Loops

This sample program demonstrates the event loop programming model. Although the callback function model is much more flexible and easy to use, you can use the event loop model in certain applications, such as modal dialog boxes. LabWindows for DOS uses the event loop model exclusively. Thus, programs translated from LabWindows for DOS to LabWindows/CVI use the event loop model unless you restructure the program to use the callback model.

16. keyfiltr.prj—Handling Keyboard Input

This sample program demonstrates how you can make a control respond to keypress events. The program displays a string control that is configured to handle keyboard input according to options selected on the main panel of this example project.

17. moustate.prj—Getting the Mouse State

This sample program demonstrates how to use the GetGlobalMouseState and GetRelativeMouseState functions. You can use these functions to determine the current position of the mouse, the current state of the mouse buttons, and the current state of the <Shift> and <Ctrl> keys on the keyboard.

18. listdelx.prj—Colors in List Boxes

This sample program demonstrates how you can add items of different colors to a list box control. Escape characters that you add to a string can control its background and foreground colors as well as the positioning in the list box. (A list of the available escape sequences appears in the help for the itemLabel parameter in the InsertListItem function panel.)

19. 2yaxis.prj - Two Y Axes on a Graph

This sample program shows how to use left and right Y axes on the same graph and how graph cursors can be assigned to the two axes.

20. intgraph.prj - Intensity Plots

This sample program demonstrates the different ways in which intensity plots can be used in a graph. It allows you to experiment with the different interpolation options as you plot a semi-random block of data.
21. autostrp.prj - Autoscaling the Y-Axis on a Strip Chart

This sample program demonstrates how you can autoscale the Y-axis on a strip chart. The User Interface Library supports autoscaling only on graphs, not on strip charts. This program simulates autoscaling by storing the data before sending it to the chart, and then using `SetAxisRange` to modify the Y-axis as needed.

22. canvas.prj - Canvas Controls

This sample program demonstrates drawing on a Canvas control, including objects such as arcs, rectangles, polygons, lines, ovals and bitmaps.

23. canvsbmk.prj - Canvas Benchmark

This sample program shows the increase in speed you can achieve by using a canvas control instead of a graph control for drawing objects such as arcs, rectangles, polygons, lines, ovals and bitmaps.

24. drawpad.prj - Using Canvas as Drawing Pad

This sample program demonstrates how to use a canvas control as a drawing port or scratch pad for the mouse.

25. piedemo.prj - Pie Chart

This sample program demonstrates how to use a canvas control to draw a pie chart using an instrument driver.

26. imagedit.prj - Changing Image Colors

This is a very simple program which modifies the colors of an existing image. It loads an image into a picture ring and then makes use of the `GetImageBits` and `SetImageBits` functions.

27. clipbord.prj - Using System Clipboard

This sample program demonstrates how to use the clipboard functions to transfer images and text to and from the system clipboard.
Appendix A
Error Conditions

This appendix lists the meanings associated with the integer error codes that the LabWindows/CVI User Interface library functions return.

Every function returns an integer code representing the result of the call. If the return code is negative, an error has occurred. Otherwise, the function successfully completed.

Note: The GetUILLErrorString function can convert an error code number into a message string.

Table A-1. User Interface Library Error Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The Interface Manager could not be opened.</td>
</tr>
<tr>
<td>-2</td>
<td>The system font could not be loaded.</td>
</tr>
<tr>
<td>-3</td>
<td>The operation attempted cannot be performed while a pop-up menu is active.</td>
</tr>
<tr>
<td>-4</td>
<td>Panel, pop-up, menu bar, or plot ID is invalid.</td>
</tr>
<tr>
<td>-5</td>
<td>Attempted to position panel at an invalid location</td>
</tr>
<tr>
<td>-6</td>
<td>Attempted to make an inoperable control the active control.</td>
</tr>
<tr>
<td>-7</td>
<td>The operation requires that a panel be loaded.</td>
</tr>
<tr>
<td>-8</td>
<td>The operation requires that a pop-up menu be active.</td>
</tr>
<tr>
<td>-9</td>
<td>The operation requires that a menu bar be loaded.</td>
</tr>
<tr>
<td>-10</td>
<td>The control is not the type expected by the function.</td>
</tr>
<tr>
<td>-11</td>
<td>Invalid menu item ID.</td>
</tr>
<tr>
<td>-12</td>
<td>Out of memory!</td>
</tr>
<tr>
<td>-13</td>
<td>Invalid control ID.</td>
</tr>
<tr>
<td>-14</td>
<td>Value is invalid or out of range.</td>
</tr>
<tr>
<td>-15</td>
<td>File is not a User Interface file or has been corrupted.</td>
</tr>
<tr>
<td>-16</td>
<td>File format is out-of-date.</td>
</tr>
</tbody>
</table>
Table A-1. User Interface Library Error Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-17</td>
<td>PCX image is corrupted or incompatible with current display type.</td>
</tr>
<tr>
<td>-18</td>
<td>No user event possible in current configuration.</td>
</tr>
<tr>
<td>-19</td>
<td>Unable to open UIR file.</td>
</tr>
<tr>
<td>-20</td>
<td>Error reading UIR file.</td>
</tr>
<tr>
<td>-21</td>
<td>Error writing UIR file.</td>
</tr>
<tr>
<td>-22</td>
<td>Error closing UIR file.</td>
</tr>
<tr>
<td>-23</td>
<td>Panel state file has invalid format.</td>
</tr>
<tr>
<td>-24</td>
<td>Invalid panel ID or menu bar ID in resource file.</td>
</tr>
<tr>
<td>-25</td>
<td>Error occurred during hardcopy output.</td>
</tr>
<tr>
<td>-26</td>
<td>Invalid default directory specified in FileSelectPopup function.</td>
</tr>
<tr>
<td>-27</td>
<td>Operation is invalid for specified object.</td>
</tr>
<tr>
<td>-28</td>
<td>Unable to find specified string in menu.</td>
</tr>
<tr>
<td>-29</td>
<td>Palette menu items can only be added to the end of the menu.</td>
</tr>
<tr>
<td>-30</td>
<td>Too many menus in the menu bar.</td>
</tr>
<tr>
<td>-31</td>
<td>Separators cannot have checkmarks.</td>
</tr>
<tr>
<td>-32</td>
<td>Separators cannot have submenus.</td>
</tr>
<tr>
<td>-33</td>
<td>The menu item must be a separator.</td>
</tr>
<tr>
<td>-34</td>
<td>The menu item cannot be a separator.</td>
</tr>
<tr>
<td>-35</td>
<td>The menu item already has a submenu.</td>
</tr>
<tr>
<td>-36</td>
<td>The menu item does not have a submenu.</td>
</tr>
<tr>
<td>-37</td>
<td>The control ID passed must be a menu ID, a menu item ID, or NULL.</td>
</tr>
<tr>
<td>-38</td>
<td>The control ID passed must be a menu ID, or a menu item ID.</td>
</tr>
<tr>
<td>-39</td>
<td>The control ID passed was not a submenu ID.</td>
</tr>
<tr>
<td>-40</td>
<td>The control ID passed was not a valid ID.</td>
</tr>
</tbody>
</table>
Table A-1. User Interface Library Error Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-41</td>
<td>The ID is not a menu bar ID.</td>
</tr>
<tr>
<td>-42</td>
<td>The ID is not a panel ID.</td>
</tr>
<tr>
<td>-43</td>
<td>This operation cannot be performed while this pop-up panel is active.</td>
</tr>
<tr>
<td>-44</td>
<td>This control/panel/menu/ does not have the specified attribute.</td>
</tr>
<tr>
<td>-45</td>
<td>The control type passed was not a valid type.</td>
</tr>
<tr>
<td>-46</td>
<td>The attribute passed is invalid.</td>
</tr>
<tr>
<td>-47</td>
<td>The fill option must be set to fill above or fill below to paint ring slide's fill color.</td>
</tr>
<tr>
<td>-48</td>
<td>The fill option must be set to fill above or fill below to paint numeric slide's fill color.</td>
</tr>
<tr>
<td>-49</td>
<td>The control passed is not a ring slide.</td>
</tr>
<tr>
<td>-50</td>
<td>The control passed is not a numeric slide.</td>
</tr>
<tr>
<td>-51</td>
<td>The control passed is not a ring slide with inc/dec arrows.</td>
</tr>
<tr>
<td>-52</td>
<td>The control passed is not a numeric slide with inc/dec arrows.</td>
</tr>
<tr>
<td>-53</td>
<td>The data type passed in is not a valid data type for the control.</td>
</tr>
<tr>
<td>-54</td>
<td>The attribute passed is not valid for the data type of the control.</td>
</tr>
<tr>
<td>-55</td>
<td>The index passed is out of range.</td>
</tr>
<tr>
<td>-56</td>
<td>There are no items in the list control.</td>
</tr>
<tr>
<td>-57</td>
<td>The buffer passed was to small for the operation.</td>
</tr>
<tr>
<td>-58</td>
<td>The control does not have a value.</td>
</tr>
<tr>
<td>-59</td>
<td>The value passed is not in the list control.</td>
</tr>
<tr>
<td>-60</td>
<td>The control passed must be a list control.</td>
</tr>
<tr>
<td>-61</td>
<td>The control passed must be a list control or a binary switch.</td>
</tr>
<tr>
<td>-62</td>
<td>The data type of the control passed must be set to a string.</td>
</tr>
<tr>
<td>-63</td>
<td>That attribute is not a settable attribute.</td>
</tr>
<tr>
<td>-64</td>
<td>The value passed is not a valid mode for this control.</td>
</tr>
</tbody>
</table>

(continues)
Table A-1. User Interface Library Error Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-65</td>
<td>A NULL pointer was passed when a non-NULL pointer was expected.</td>
</tr>
<tr>
<td>-66</td>
<td>The text background color on a menu ring cannot be set or gotten.</td>
</tr>
<tr>
<td>-67</td>
<td>The ring control passed must be one of the menu ring styles.</td>
</tr>
<tr>
<td>-68</td>
<td>Text cannot be colored transparent.</td>
</tr>
<tr>
<td>-69</td>
<td>A value cannot be converted to the specified data type.</td>
</tr>
<tr>
<td>-70</td>
<td>Invalid tab order position for control.</td>
</tr>
<tr>
<td>-71</td>
<td>The tab order position of an indicator-only control cannot be set.</td>
</tr>
<tr>
<td>-72</td>
<td>Invalid number.</td>
</tr>
<tr>
<td>-73</td>
<td>There is no menu bar installed for the panel.</td>
</tr>
<tr>
<td>-74</td>
<td>The control passed is not a text box.</td>
</tr>
<tr>
<td>-75</td>
<td>Invalid scroll mode for chart.</td>
</tr>
<tr>
<td>-76</td>
<td>Invalid image mode for picture.</td>
</tr>
<tr>
<td>-77</td>
<td>The attribute is valid for child panels only. Some attributes of top level panels are determined by the host operating system.</td>
</tr>
<tr>
<td>-78</td>
<td>The list control passed is not in check mode.</td>
</tr>
<tr>
<td>-79</td>
<td>The control values could not be completely loaded into the panel because the panel has changed.</td>
</tr>
<tr>
<td>-80</td>
<td>Maximum value must be greater than minimum value.</td>
</tr>
<tr>
<td>-81</td>
<td>Graph does not have that many cursors.</td>
</tr>
<tr>
<td>-82</td>
<td>Invalid plot.</td>
</tr>
<tr>
<td>-83</td>
<td>New cursor position is outside plot area.</td>
</tr>
<tr>
<td>-84</td>
<td>The length of the string exceeds the limit.</td>
</tr>
<tr>
<td>-85</td>
<td>The specified callback function does not have the required prototype.</td>
</tr>
<tr>
<td>-86</td>
<td>The specified callback function is not a known function.</td>
</tr>
</tbody>
</table>
Table A-1. User Interface Library Error Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-87</td>
<td>Graph cannot be in this mode without cursors.</td>
</tr>
<tr>
<td>-88</td>
<td>Invalid axis scaling mode for chart.</td>
</tr>
<tr>
<td>-89</td>
<td>The font passed is not in font table.</td>
</tr>
<tr>
<td>-90</td>
<td>The attribute value passed is not valid.</td>
</tr>
<tr>
<td>-91</td>
<td>Too many files are open.</td>
</tr>
<tr>
<td>-92</td>
<td>Unexpectedly reached end of file.</td>
</tr>
<tr>
<td>-93</td>
<td>Input/Output error.</td>
</tr>
<tr>
<td>-94</td>
<td>File not found.</td>
</tr>
<tr>
<td>-95</td>
<td>File access permission denied.</td>
</tr>
<tr>
<td>-96</td>
<td>File access is not enabled.</td>
</tr>
<tr>
<td>-97</td>
<td>Disk is full.</td>
</tr>
<tr>
<td>-98</td>
<td>File already exists.</td>
</tr>
<tr>
<td>-99</td>
<td>File already open.</td>
</tr>
<tr>
<td>-100</td>
<td>Badly formed pathname.</td>
</tr>
<tr>
<td>-101</td>
<td>File is damaged.</td>
</tr>
<tr>
<td>-102</td>
<td>The format of the resource file is too old to read.</td>
</tr>
<tr>
<td>-103</td>
<td>File is corrupted.</td>
</tr>
<tr>
<td>-104</td>
<td>The operation could not be performed.</td>
</tr>
<tr>
<td>-105</td>
<td>The control passed is not a ring knob, dial, or gauge.</td>
</tr>
<tr>
<td>-106</td>
<td>The control passed is not a numeric knob, dial, or gauge.</td>
</tr>
<tr>
<td>-107</td>
<td>The count passed is out of range.</td>
</tr>
<tr>
<td>-108</td>
<td>The keycode is not valid.</td>
</tr>
<tr>
<td>-109</td>
<td>The picture control has no image.</td>
</tr>
<tr>
<td>-110</td>
<td>Panel background cannot be colored transparent.</td>
</tr>
</tbody>
</table>

(continues)
Table A-1. User Interface Library Error Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>-111</td>
<td>Title background cannot be colored transparent.</td>
</tr>
<tr>
<td>-112</td>
<td>Not enough memory for printing.</td>
</tr>
<tr>
<td>-113</td>
<td>The shortcut key passed is reserved.</td>
</tr>
<tr>
<td>-114</td>
<td>The format of the file is newer than this version of CVI.</td>
</tr>
<tr>
<td>-115</td>
<td>System printing error.</td>
</tr>
<tr>
<td>-116</td>
<td>Driver printing error.</td>
</tr>
<tr>
<td>-117</td>
<td>The deferred callback queue is full.</td>
</tr>
<tr>
<td>-118</td>
<td>The mouse cursor passed is invalid.</td>
</tr>
<tr>
<td>-119</td>
<td>Printing functions are not reentrant.</td>
</tr>
<tr>
<td>-120</td>
<td>Out of Windows GDI space.</td>
</tr>
<tr>
<td>-121</td>
<td>The panel must be visible.</td>
</tr>
<tr>
<td>-122</td>
<td>The control must be visible.</td>
</tr>
<tr>
<td>-123</td>
<td>The attribute is not valid for the type of plot.</td>
</tr>
<tr>
<td>-124</td>
<td>Intensity plots cannot use transparent colors.</td>
</tr>
<tr>
<td>-125</td>
<td>Color is invalid.</td>
</tr>
<tr>
<td>-126</td>
<td>The specified callback function differs only by a leading underscore from another function or variable. Change one of the names for proper linking.</td>
</tr>
<tr>
<td>-127</td>
<td>Bitmap is invalid.</td>
</tr>
<tr>
<td>-128</td>
<td>There is no image in the control.</td>
</tr>
</tbody>
</table>
Appendix B
Customer Communication

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

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

Electronic Services

Bulletin Board Support

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

- United States: (512) 794-5422 or (800) 327-3077
  Up to 14,400 baud, 8 data bits, 1 stop bit, no parity
- United Kingdom: 01635 551422
  Up to 9,600 baud, 8 data bits, 1 stop bit, no parity
- France: 1 48 65 15 59
  Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FaxBack Support

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

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

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

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

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

Fax and Telephone Support

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

<table>
<thead>
<tr>
<th>Country</th>
<th>Telephone</th>
<th>Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>03 9 879 9422</td>
<td>03 9 879 9179</td>
</tr>
<tr>
<td>Austria</td>
<td>0662 45 79 90 0</td>
<td>0662 45 79 90 19</td>
</tr>
<tr>
<td>Belgium</td>
<td>02 757 00 20</td>
<td>02 757 03 11</td>
</tr>
<tr>
<td>Canada (Ontario)</td>
<td>519 622 9310</td>
<td></td>
</tr>
<tr>
<td>Canada (Quebec)</td>
<td>514 694 8521</td>
<td>514 694 4399</td>
</tr>
<tr>
<td>Denmark</td>
<td>45 76 26 00</td>
<td>45 76 26 02</td>
</tr>
<tr>
<td>Finland</td>
<td>90 527 2321</td>
<td>90 502 2930</td>
</tr>
<tr>
<td>France</td>
<td>1 48 14 24 24</td>
<td>1 48 14 24 14</td>
</tr>
<tr>
<td>Germany</td>
<td>089 741 31 30</td>
<td>089 714 60 35</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>2645 3186</td>
<td>2686 8505</td>
</tr>
<tr>
<td>Italy</td>
<td>02 413091</td>
<td>02 41309215</td>
</tr>
<tr>
<td>Japan</td>
<td>03 5472 2970</td>
<td>03 5472 2977</td>
</tr>
<tr>
<td>Korea</td>
<td>02 596 7456</td>
<td>02 596 7455</td>
</tr>
<tr>
<td>Mexico</td>
<td>95 800 010 0793</td>
<td>5 520 3282</td>
</tr>
<tr>
<td>Netherlands</td>
<td>0348 433466</td>
<td>0348 430673</td>
</tr>
<tr>
<td>Norway</td>
<td>32 84 84 00</td>
<td>32 84 86 00</td>
</tr>
<tr>
<td>Singapore</td>
<td>2265886</td>
<td>2265887</td>
</tr>
<tr>
<td>Spain</td>
<td>91 640 0085</td>
<td>91 640 0533</td>
</tr>
<tr>
<td>Sweden</td>
<td>08 730 49 70</td>
<td>08 730 43 70</td>
</tr>
<tr>
<td>Switzerland</td>
<td>056 200 51 51</td>
<td>056 200 51 55</td>
</tr>
<tr>
<td>Taiwan</td>
<td>02 377 1200</td>
<td>02 737 4644</td>
</tr>
<tr>
<td>U.K.</td>
<td>01635 523545</td>
<td>01635 523154</td>
</tr>
</tbody>
</table>
Technical Support Form

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

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

Name_______________________________________________________________________________________
Company____________________________________________________________________________________
Address _____________________________________________________________________________________

Fax (_____)_________________________ Phone (_____)___________________________

Computer brand ___________ Model ___________ Processor ___________

Operating system:    Windows 3.1,    Windows for Workgroups 3.11,    Windows NT 3.1,    Windows NT 3.5,    Windows 95,    other (include version number)______________________________

Version of Excel (look at Excel’s About box):  5.0, 5.0c, other______________________________

Clock Speed ________MHz       RAM ________MB       Display adapter ___________

Mouse ___yes ___no       Other adapters installed ___________

Hard disk capacity _____MB       Brand ___________

Instruments used ___________

National Instruments hardware product model ___________ Revision ___________

Configuration ___________

National Instruments software product ___________ Version ___________

Configuration ___________

The problem is ___________

________________________________________________________________________

List any error messages ____________________________________________________________________

________________________________________________________________________

________________________________________________________________________

The following steps will reproduce the problem ____________________________________________________________________

________________________________________________________________________
Hardware and Software Configuration Form

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

National Instruments Products

Data Acquisition Hardware Revision ______________________________________________________________
Interrupt Level of Hardware __________________________________________________________________
DMA Channels of Hardware ____________________________________________________________________
Base I/O Address of Hardware __________________________________________________________________
NI-DAQ, LabVIEW, or LabWindows Version _________________________________________________________________________

Other Products

Computer Make and Model _____________________________________________________________________
Microprocessor ______________________________________________________________________________
Clock Frequency _____________________________________________________________________________
Type of Video Board Installed ___________________________________________________________________
Operating System _____________________________________________________________________________
Operating System Version ______________________________________________________________________
Operating System Mode _______________________________________________________________________
Programming Language ________________________________________________________________________
Programming Language Version ___________________________________________________________________
Other Boards in System _________________________________________________________________________
Base I/O Address of Other Boards ________________________________________________________________
DMA Channels of Other Boards _________________________________________________________________
Interrupt Level of Other Boards __________________________________________________________________
Documentation Comment Form

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

Title: **LabWindows®/CVI User Interface Reference Manual**
Edition Date: **July 1996**
Part Number: **320683C-01**

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

___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________

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

___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________
___________________________________________________________________________________________

Thank you for your help.

Name_______________________________________________________________________________________
Title________________________________________________________________________________________
Company____________________________________________________________________________________
Address _____________________________________________________________________________________

Fax (______)________________________ Phone (______)________________________

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX  78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678
Glossary

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>m-</td>
<td>milli-</td>
<td>10^-3</td>
</tr>
<tr>
<td>µ-</td>
<td>micro-</td>
<td>10^-6</td>
</tr>
</tbody>
</table>

**B**

binary switch  A control that selects between two states: on and off.

bitmap  A set of data that can be used to draw a graphic image. The data consist of information determining the height and width of the image or pixel grid, and the color of each pixel.

bps  Bits per second.

**C**

canvas  An arbitrary drawing surface to display text, shapes, and bitmap images.

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.

confirm pop-up panel  Allows you to confirm an action before it is taken.

control  An object that resides on a panel and provides a mechanisms for accepting input from and displaying information to the user.

**E**

event  Informs the application program that the user has performed an action. An event is generated whenever the user selects a command from the menu bar or manipulates a control that was configured to generate events.
Glossary

F

file select pop-up panel A predefined pop-up panel that displays a list of files on disk from which the user can select.

G

graph control Displays graphical data as one or more plots.

graph pop-up panel A predefined pop-up panel for displaying numerical data graphically. There are different functions for graphing X, Y, X-Y, and waveform data sets.

H

hot control Similar to normal control except that the control generates an event when acted upon by the user. Events are returned to the application program, which must determine what action to take. Normally, a hot control generates an event when its state is changed. For example, if the user moves a binary switch from off to on, an event is generated.

I

immediate command A menu title that ends in an exclamation point does not have an associated menu. Selecting an immediate command executes it directly.

in inches.

indicator control A control that can be changed programmatically but cannot be operated by the user. LED, scale, text, text box, graph (without cursors), and strip chart controls are always indicators.

L

LED A control that is modeled to operate like light emitting diodes, which indicate on/off states. When an LED is on, it appears lighted.
M

MB  Megabytes of memory.

menu bar  A mechanism for encapsulating a set of commands. A menu bar appears at the top of the screen and contains a set of menu titles.

message pop-up panel  A predefined pop-up panel for displaying a message.

N

normal control  A control that can be operated by the user and changed programmatically.

numeric/string control  Used to input or view numeric values or text strings. A typical use of this control might be to input a person's name or to display a voltage value.

P

panel  A rectangular region of the screen containing a set of controls that accept input from the user and display information to the user. Panels can perform many different functions, from representing the front panel of an instrument to allowing the user to select a file name.

pen  A drawing construct which defines the characteristics to be used to draw images on a canvas control. The settable attributes include width, style, color, mode and pattern of the line or object drawn.

pixel  An element of a picture. The smallest resolvable rectangular area of an image, either on a screen or stored in memory. Each pixel has its own brightness and color, usually represented as a red, green, and blue intensities (see RGB).

point  A structure used to specify the location of a point in the Cartesian coordinate systems used in canvas controls and bitmaps. The structure contains two integer values, $x$ and $y$.

plot  Consists of a curve, a point, a geometric shape, or a text string.

pop-up panel  A panel that pops up, accepts user input, and then disappears.

prompt pop-up panel  A predefined pop-up panel for requesting input from the user.
pull-down menu  A menu title without an exclamation point contains a collection of commands that appear when you select it.

push button  Used to trigger an action indicated by a label on the button.

R

radio button  Similar to binary switches, these buttons allow you to switch between the off and on state. A radio button is either pressed in or popped out.

rect  A structure used to specify the location and size of a rectangle in the Cartesian coordinate systems used in canvas controls and bitmaps. The structure contains four integer values, top, left, height, and width.

resource file  Contains all of the object associated with a user interface. This includes menu bars, panels, controls, pop-up panels, preferences, images, and fonts. To display user interface objects, an application program must call the User Interface Library to load them from the resource file. A single application program can use multiple resource files.

RGB  Red-green-blue. The three colors of light which can be mixed to produce any other color.

ring control  Allows you to select from a group of items. Only the currently selected item shows. You can scroll forward or backward through the list of items, or you can select an item through its pop-up format.

S

s  Seconds.

scale control  Indicates the magnitude of a value relative to a predefined range. The scale moves up and down within the control as its value changes. Scale controls are indicator controls and cannot be operated.

selection list control  Used to select a item from a list.

slide control  Allows you to select one item from a group of items. The slider, or cross-bar, indicates the current selection.
slider

The cross-bar of the slide control that points to the currently selected item.

standard libraries


string control

See numeric/string control.

strip chart control

A graph that displays graphical data as one or more traces in real time.

stylized fonts

Fixed-size, bitmapped fonts that have a variety of type faces.

system fonts

Bitmapped fonts designed for a particular display adapter. They are sized to fit the standard 80-column by 25-line display format. The User Interface Library automatically selects the appropriate system font for your adapter.

T

text boxes

Display a window of text.

text controls

Display a string of text.

timer control

A user interface control that schedules the periodic execution of a callback function. A typical use of this control might be to update a graph every second.

traces

Curves in strip charts.

U

User Interface Editor

Used to create resource files.

V

validate control

Similar to hot control except that all numeric/scalar controls on the panel are validated before the event is generated. The value of each numeric/scalar control is checked against its predefined range. If an invalid condition is found, a dialog box appears to inform you.
Index

A

Add File to Project command, File menu, 2-4
Align Horizontal Centers command, Arrange menu, 2-21
Alignment command, Arrange menu, 2-20 to 2-21
All Callbacks command, Generate menu, 2-27
All Code command description, 2-25 to 2-26
Generate All Code dialog box, 2-25
AllocBitmapData function, 4-12 to 4-13
AllocImageBits function, 4-14 to 4-15
Always Append Code to End option, Preferences command, 2-30
Apply Default Font command, Edit menu, 2-15
Arrange menu, User Interface Editor
  Align Horizontal Centers command, 2-21
  Alignment command, 2-20 to 2-21
  Center Label command, 2-22
  Control Coordinates command, 2-23
  Control ZPlane Order command, 2-22
  Distribute Vertical Centers command, 2-22
  Distribution command, 2-21 to 2-22 illustration, 2-20
ASCII keys, 3-26, 4-157 to 4-158
ASCII text format
  loading objects into User Interface Editor window, 2-34
  saving contents of User Interface Editor window in, 2-33
Assign Missing Constants command, Options menu, 2-33
ATTR_ACTIVATE_WHEN_CLICKED_ON, 3-12
ATTR_ACTIVE, 3-12
ATTR_ACTIVE_YAXIS, 3-72
ATTR.Allow_MISSING_CALLBACKS, 3-86
ATTR_ALLOW_ROOM_FOR_IMAGES, 3-39
ATTR_ALLOW_UNSAFE_TIMER_EVENTS, 3-86
ATTR_BACKCOLOR, 3-12
ATTR_BINARY_SWITCH_COLOR, 3-41
ATTR_BORDER_VISIBLE, 3-68
ATTR_CALLBACK_DATA, 3-12, 3-24, 3-32
ATTR_CALLBACK_FUNCTION_POINTER, 3-12, 3-24, 3-32
ATTR_CALLBACK_NAME, 3-12, 3-24, 3-32
ATTR_CALLBACK_NAME_LENGTH, 3-12, 3-24, 3-32
ATTR_CAN_MAXIMIZE, 3-12, 3-17
ATTR_CAN_MINIMIZE, 3-12, 3-17
ATTR_CHECK_MODE, 3-39
ATTR_CHECK_RANGE, 3-37, 3-51
ATTR_CHECK_STYLE, 3-40
ATTR_CHECKED, 3-25
ATTR_CLOSE_CTRL, 3-12
ATTR_CLOSE_ITEM_VISIBLE, 3-12, 3-17
ATTR_CMD_BUTTON_COLOR, 3-40
ATTR_COLOR_MODE, 3-89, 3-91
ATTR_CONFORM_TO_SYSTEM, 3-13
ATTR_CONSTANT_NAME, 3-13, 3-24, 3-32
ATTR_CONSTANT_NAME_LENGTH, 3-13, 3-24, 3-32
ATTR_COPY_ORIGINAL_DATA, 3-72, 3-84 to 3-85
ATTR.CROSS_HAIR_STYLE, 3-74, 3-78
ATTR_CTRL_INDEX, 3-37
ATTR_CTRL_MODE, 3-33
ATTR_CTRL_STYLE, 3-32, 3-46 to 3-50
ATTR_CTRL_TAB_POSITION, 3-33
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_CTRL_VAL</td>
<td>3-33</td>
</tr>
<tr>
<td>ATTR_CURSOR_COLOR</td>
<td>3-74</td>
</tr>
<tr>
<td>ATTR_CURSOR_MODE</td>
<td>3-74</td>
</tr>
<tr>
<td>ATTR_CURSOR_POINT_STYLE</td>
<td>3-74, 3-78 to 3-79</td>
</tr>
<tr>
<td>ATTR_CURSOR_YAXIS</td>
<td>3-74</td>
</tr>
<tr>
<td>ATTR_DATA_MODE</td>
<td>3-72, 3-84</td>
</tr>
<tr>
<td>ATTR_DATA_TYPE</td>
<td>3-36, 3-50</td>
</tr>
<tr>
<td>ATTR_DFLT_INDEX</td>
<td>3-37</td>
</tr>
<tr>
<td>ATTR_DFLT_VALUE</td>
<td>3-37</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_HEIGHT</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_LEFT</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_TOP</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_DIG_DISP_WIDTH</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_DIMMED</td>
<td>3-13, 3-24, 3-32</td>
</tr>
<tr>
<td>ATTR_DRAW_LIGHT_BEVEL</td>
<td>3-25</td>
</tr>
<tr>
<td>ATTR_DRAW_POLICY</td>
<td>3-56, 3-57</td>
</tr>
<tr>
<td>ATTR_DUPLEX</td>
<td>3-89, 3-90</td>
</tr>
<tr>
<td>ATTR_EDGE_STYLE</td>
<td>3-68</td>
</tr>
<tr>
<td>ATTR_EJECT_AFTER</td>
<td>3-89, 3-90</td>
</tr>
<tr>
<td>ATTR_ENABLE_ZOOMING</td>
<td>3-72</td>
</tr>
<tr>
<td>ATTR_ENABLED</td>
<td>3-44, 3-64, 3-65</td>
</tr>
<tr>
<td>ATTR_ENTER_IS_NEWLINE</td>
<td>3-38</td>
</tr>
<tr>
<td>ATTR_EXTRA_LINES</td>
<td>3-38</td>
</tr>
<tr>
<td>ATTR_FILL_COLOR</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_FILL_HOUSING_COLOR</td>
<td>3-43</td>
</tr>
<tr>
<td>ATTR_FILL_OPTION</td>
<td>3-43</td>
</tr>
<tr>
<td>ATTR_FIRST_CHILD</td>
<td>3-13</td>
</tr>
<tr>
<td>ATTR_FIRST_VISIBLE_LINE</td>
<td>3-39</td>
</tr>
<tr>
<td>ATTR_FIX_MODE</td>
<td>3-44</td>
</tr>
<tr>
<td>ATTR_FLOATING</td>
<td>3-13</td>
</tr>
<tr>
<td>ATTR_FORMAT</td>
<td>3-37, 3-51</td>
</tr>
<tr>
<td>ATTR_FRAME_COLOR</td>
<td>3-13, 3-34</td>
</tr>
<tr>
<td>ATTR_FRAME_STYLE</td>
<td>3-13, 3-18 to 3-19</td>
</tr>
<tr>
<td>ATTR_FRAME_THICKNESS</td>
<td>3-13</td>
</tr>
<tr>
<td>ATTR_FRAME_VISIBLE</td>
<td>3-44</td>
</tr>
<tr>
<td>ATTR_GRAPH_BGCOLOR</td>
<td>3-68</td>
</tr>
<tr>
<td>ATTR_GRID_COLOR</td>
<td>3-68</td>
</tr>
<tr>
<td>ATTR_HEIGHT</td>
<td>3-13, 3-33</td>
</tr>
<tr>
<td>ATTR_HILITE_CURRENT_ITEM</td>
<td>3-40</td>
</tr>
<tr>
<td>ATTR_HSCROLL_OFFSET</td>
<td>3-13, 3-38</td>
</tr>
<tr>
<td>ATTR_HSCROLL_OFFSET_MAX</td>
<td>3-13</td>
</tr>
<tr>
<td>ATTR_IMAGE_FILE</td>
<td>3-45</td>
</tr>
<tr>
<td>ATTR_IMAGE_FILE_LENGTH</td>
<td>3-45</td>
</tr>
<tr>
<td>ATTR_INCR_VALUE</td>
<td>3-37</td>
</tr>
<tr>
<td>ATTR_INNER_LOG_MARKERS_VISIBLE</td>
<td>3-68</td>
</tr>
<tr>
<td>ATTR_INTERPOLATE_PIXELS</td>
<td>3-75</td>
</tr>
<tr>
<td>ATTR_INTERVAL</td>
<td>3-44, 3-64</td>
</tr>
<tr>
<td>ATTR_ITEM_NAME</td>
<td>3-25</td>
</tr>
<tr>
<td>ATTR_ITEM_NAME_LENGTH</td>
<td>3-25</td>
</tr>
<tr>
<td>ATTR_LABEL_BGCOLOR</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_BOLD</td>
<td>3-34</td>
</tr>
<tr>
<td>ATTR_LABEL_COLOR</td>
<td>3-34</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT</td>
<td>3-34</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT_NAME_LENGTH</td>
<td>3-34</td>
</tr>
<tr>
<td>ATTR_LABEL_HEIGHT</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_HEIGHT_LENGTH</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_BOLD</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT_NAME_LENGTH</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT_NAME_LENGTH</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT_STYLE</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT_UNDERLINE</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_FONT_VISIBLE</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_HEIGHT</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_HEIGHT_LENGTH</td>
<td>3-35</td>
</tr>
<tr>
<td>ATTR_LABEL_LEFT</td>
<td>3-14, 3-33</td>
</tr>
<tr>
<td>ATTR_LINE_STYLE</td>
<td>3-74, 3-79</td>
</tr>
<tr>
<td>ATTR_MARKER_END_ANGLE</td>
<td>3-43</td>
</tr>
<tr>
<td>ATTR_MARKER_START_ANGLE</td>
<td>3-43</td>
</tr>
<tr>
<td>ATTR_MARKER_STYLE</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_MAX_ENTRY_LENGTH</td>
<td>3-38</td>
</tr>
<tr>
<td>ATTR_MAX_VALUE</td>
<td>3-37</td>
</tr>
<tr>
<td>ATTR_MENU_ARROW_COLOR</td>
<td>3-43</td>
</tr>
<tr>
<td>ATTR_MENU_BAR_VISIBLE</td>
<td>3-14</td>
</tr>
<tr>
<td>ATTR_MENU_HEIGHT</td>
<td>3-14</td>
</tr>
<tr>
<td>ATTR_MENU_NAME</td>
<td>3-25</td>
</tr>
<tr>
<td>ATTR_MENU_NAME_LENGTH</td>
<td>3-25</td>
</tr>
<tr>
<td>ATTR_MIN_VALUE</td>
<td>3-37</td>
</tr>
<tr>
<td>ATTR_MOUSE_CURSOR</td>
<td>3-14, 3-19</td>
</tr>
<tr>
<td>ATTR_MOVABLE</td>
<td>3-14</td>
</tr>
<tr>
<td>ATTR_NEEDLE_COLOR</td>
<td>3-42</td>
</tr>
<tr>
<td>ATTR_NEXT_CTRL</td>
<td>3-33</td>
</tr>
<tr>
<td>ATTR_NEXT_PANEL</td>
<td>3-14</td>
</tr>
<tr>
<td>ATTR_NO_EDIT_TEXT</td>
<td>3-40</td>
</tr>
</tbody>
</table>
ATTR_NSCELL_OFFSET_MAX, 3-37
ATTR_NUM_CHILDREN, 3-14
ATTR_NUM_COPIES, 3-89
ATTR_NUM_CTRLS, 3-14
ATTR_NUM_CURSOR, 3-72
ATTR_NUM_MENU_ITEMS, 3-25
ATTR_NUM_MENUS, 3-25
ATTR_NUM_POINTS, 3-75
ATTR_NUM_TRACES, 3-73
ATTR_OFF_COLOR, 3-41
ATTR_OFF_TEXT, 3-41
ATTR_OFF_TEXT_LENGTH, 3-41
ATTR_OFF_VALUE, 3-41
ATTR_OFF_VALUE_LENGTH, 3-41
ATTR_ON_COLOR, 3-41
ATTR_ON_TEXT, 3-41
ATTR_ON_TEXT_LENGTH, 3-41
ATTR_ON_VALUE, 3-41
ATTR_ON_VALUE_LENGTH, 3-41
ATTR_ORIENTATION, 3-89, 3-90
ATTR_OVERLAPPED, 3-33
ATTR_OVERLAPPED_POLICY, 3-56, 3-58
ATTR_PANEL_FIRST_CTRL, 3-14
ATTR_PANEL_MENU_BAR_CONSTANT, 3-14
ATTR_PANEL_MENU_BAR_CONSTANT_LENGTH, 3-14
ATTR_PANEL_PARENT, 3-14
ATTR_PAPER_HEIGHT, 3-89, 3-90
ATTR_PAPER_WIDTH, 3-89, 3-90
ATTR_PARENT_SHARES_SHORTCUT_KEYS, 3-14
ATTR_PEN_COLOR, 3-56
ATTR_PEN_FILL_COLOR, 3-56
ATTR_PEN_MODE, 3-56, 3-58
ATTR_PEN_PATTERN, 3-56, 3-58 to 3-59
ATTR_PEN_STYLE, 3-56
ATTR_PEN_WIDTH, 3-57
ATTR_PICT_BGCOLOR, 3-44
ATTR_PLOT_AREA_HEIGHT, 3-68
ATTR_PLOT_AREA_WIDTH, 3-68
ATTR_PLOT_BGCOLOR, 3-68
ATTR_PLOT_FONT, 3-75
ATTR_PLOT_FONT_NAME_LENGTH, 3-75
ATTR_PLOT_ORIGIN, 3-75, 3-81
ATTR_PLOT_SNAPPABLE, 3-81
ATTR_PLOT_STYLE, 3-74, 3-80
ATTR_PLOT_XDATA, 3-76
ATTR_PLOT_XDATA_SIZE, 3-77
ATTR_PLOT_XDATA_TYPE, 3-76
ATTR_PLOT_YAXIS, 3-75
ATTR_PLOT_YDATA, 3-76
ATTR_PLOT_YDATA_SIZE, 3-77
ATTR_PLOT_YDATA_TYPE, 3-76
ATTR_PLOT_ZDATA, 3-76
ATTR_PLOT_ZDATA_SIZE, 3-77
ATTR_PLOT_ZDATA_TYPE, 3-76
ATTR_PLOT_ZPLANE_POSITION, 3-76
ATTR_POINTS_PER_SCREEN, 3-73
ATTR_PRECISION, 3-37
ATTR_REFRESH_GRAPH, 3-72, 3-83
ATTR_REPORT_LOAD_FAILURE, 3-86, 3-87
ATTR_SCROLL_BAR_COLOR, 3-14, 3-39
ATTR_SCROLL_BAR_SIZE, 3-39
ATTR_SCROLL_BARS, 3-15, 3-39
ATTR_SCROLL_MODE, 3-76
ATTR_SHORTCUT_KEY, 3-25, 3-40
ATTR_SHOW_DIG_DISP, 3-42
ATTR_SHOW_INCDEC_ARROWS, 3-37
ATTR_SHOW_MORE_BUTTON, 3-43
ATTR_SHOW_RADIX, 3-37
ATTR_SHOW_TRANSPARENT, 3-43
ATTR_SIZABLE, 3-15
ATTR_SIZE_TO_TEXT, 3-40
ATTR_SLIDER_COLOR, 3-42
ATTR_SLIDER_HEIGHT, 3-43
ATTR_SLIDER_WIDTH, 3-43
ATTR_SMOOTH_UPDATE, 3-73, 3-82, 3-84
ATTR_STRING_TEXT_LENGTH, 3-38
ATTR_SUBIMAGE_HEIGHT, 3-45
ATTR_SUBIMAGE_WIDTH, 3-45
ATTR_SUBMENU_ID, 3-25
ATTR_SYSTEM_MENU_VISIBLE, 3-15
ATTR_SYSTEM_WINDOW_HANDLE, 3-15, 3-17
ATTR_TAB_INTERVAL, 3-89
ATTR_TEXT_BGCOLOR, 3-36
ATTR_TEXT_BOLD, 3-35
ATTR_TEXT_CLICK_TOGGLES_CHECK, 3-39, 3-40
ATTR_TEXT_COLOR, 3-35
ATTR_TEXT_FONT, 3-35
ATTR_TEXT_FONT_NAME_LENGTH, 3-36
ATTR_TEXT_ITALIC, 3-36
ATTR_TEXT_JUSTIFY, 3-36
ATTR_TEXT_POINT_SIZE, 3-36
ATTR_TEXT_RAISED, 3-40
ATTR_TEXT_SELECTION_LENGTH, 3-38
ATTR_TEXT_SELECTION_START, 3-38
ATTR_TEXT_STRIKEOUT, 3-36
ATTR_TEXT_UNDERLINE, 3-36
ATTR_TEXT_WRAP, 3-89
ATTR_TICK_STYLE, 3-42
ATTR_TITLE, 3-15
ATTR_TITLE_BACKCOLOR, 3-15
ATTR_TITLE_BOLD, 3-15
ATTR_TITLE_COLOR, 3-15
ATTR_TITLE_FONT, 3-15
ATTR_TITLE_FONT_NAME_LENGTH, 3-15
ATTR_TITLE.Italic, 3-15
ATTR_TITLE_LENGTH, 3-15
ATTR_TITLE_POINT_SIZE, 3-15
ATTR_TITLE_SIZE_TO_FONT, 3-16
ATTR_TITLE_strikeout, 3-16
ATTR_TITLE_UNDERLINE, 3-16
ATTR_TITLEBAR_THICKNESS, 3-16
ATTR_TITLEBAR_VISIBLE, 3-16
ATTR_TOP, 3-16, 3-33
ATTR_TOTAL_LINES, 3-38
ATTR_TRACE_BGCOLOR, 3-76
ATTR_TRACE_COLOR, 3-74
ATTR_TRACE_POINT_STYLE, 3-74, 3-78 to 3-79
ATTR_TRACE_VISIBLE, 3-74
ATTR_USE_SUBIMAGE, 3-45
ATTR_VISIBLE, 3-16, 3-33
ATTR_VISIBLE_LINES, 3-39
ATTR_VSCROLL_OFFSET, 3-16
ATTR_VSCROLL_OFFSET_MAX, 3-16
ATTR_WIDTH, 3-16, 3-33
ATTR_WINDOW_ZOOM, 3-16, 3-17
ATTR_WRAP_MODE, 3-38
ATTR_XAXIS_GAIN, 3-69
ATTR_XAXIS_OFFSET, 3-69
ATTR_XCOORD_AT_ORIGIN, 3-57
ATTR_XDIVISIONS, 3-69
ATTR_XENG_UNITS, 3-69
ATTR_XFORMAT, 3-69
ATTR_XGRID_VISIBLE, 3-69
ATTR_XLABEL_VISIBLE, 3-69
ATTR_XMAP_MODE, 3-73
ATTR_XMARK_ORIGIN, 3-73
ATTR_XNAME, 3-69
ATTR_XNAME_LENGTH, 3-70
ATTR_XOFFSET, 3-89, 3-90
ATTR_XPRECISION, 3-70
ATTR_XRESOLUTION, 3-89, 3-90
ATTR_XREVERSE, 3-73
ATTR_XSCALING, 3-57
ATTR_XUSE_LABEL_STRINGS, 3-69
ATTR_XLABEL_BOLD, 3-70
ATTR_XLABEL_COLOR, 3-70
ATTR_XLABEL_FONT, 3-70
ATTR_XLABEL_FONT_NAME_LENGTH, 3-70
ATTR_XLABEL_ITALIC, 3-70
ATTR_XLABEL_POINT_SIZE, 3-70
ATTR_XLABEL_STRIKEOUT, 3-70
ATTR_XLABEL_UNDERLINE, 3-70
ATTR_XMARK_ORIGIN, 3-73
ATTR_XNAME, 3-71
ATTR_XNAME_POINT_SIZE, 3-71
ATTR_XNAME_STRIKEOUT, 3-71
ATTR_XNAME_UNDERLINE, 3-71
ATTR_YAXIS_GAIN, 3-69
ATTR_YAXIS_OFFSET, 3-70
ATTR_YAXIS_REVERSE, 3-70
ATTR_YCOORD_AT_ORIGIN, 3-57
Index

ATTR_YDIVISIONS, 3-71
ATTR_YENG_UNITS, 3-71
ATTR_YFORMAT, 3-71
ATTR_YGRID_VISIBLE, 3-72
ATTR_YLABEL_VISIBLE, 3-72
ATTR_YMAP_MODE, 3-72
ATTR_YMARK_ORIGIN, 3-73
ATTR_YNAME, 3-72
ATTR_YNAME_LENGTH, 3-72
ATTR_YOFFSET, 3-89, 3-90
ATTR_YPRECISION, 3-72
ATTR_YRESOLUTION, 3-89, 3-90
ATTR_YSCALING, 3-57
ATTR_YUSE_LABEL_STRINGS, 3-70
ATTR_ZPLANE_POSITION, 3-16, 3-33
attributes
canvas controls, 3-56 to 3-59
  ATTR_DRAW_POLICY values
    (table), 3-57
  ATTR_OVERLAPPED_POLICY
    values (table), 3-58
  ATTR_PEN_MODE values
    (table), 3-58
  list of attributes (table), 3-56 to 3-57
  pixel values for
    ATTR_PEN_PATTERN, 3-58 to 3-59
controls
  ATTR_CHECK_RANGE values
    (table), 3-51
control data types for
  ATTR_DATA_TYPE (table), 3-50
control styles for
  ATTR_CTRL_STYLE (table), 3-46 to 3-50
  list of attributes (table), 3-32 to 3-33
  numeric formats for
    ATTR_FORMAT (table), 3-51
graph and strip chart controls
cursor styles for
  ATTR_CROSSHAIR_STYLE
    (table), 3-78
discussion of specific attributes, 3-77 to 3-80
  line styles for ATTR_LINE_STYLE
    (table), 3-79
list of attributes (table), 3-68 to 3-77
plot origins, 3-80 to 3-81
plot styles for ATTR_PLOT_STYLE
  (table), 3-80
styles for
  ATTR_CURSOR_POINT_STYLE
  and
  ATTR_TRACE_POINT_STYLE
  (table), 3-78 to 3-79
two Y axes, 3-81
values for ATTR_PLOT_ORIGIN
  (table), 3-81
hard copy, 3-89
menu bars
  constants for masking three bit fields
    (table), 3-28
  menu and menu item attributes
    (table), 3-24 to 3-25
  modifiers and virtual keys for
    shortcut keys (table), 3-26 to 3-27
panels
  color values (table), 3-17 to 3-18
  fonts, 3-20 to 3-21
  frame style values, 3-18
  geometric attributes (table), 3-18
  list of attributes (table), 3-12 to 3-16
  values and cursor styles for
    ATTR_MOUSE-CURSOR
    (table), 3-19
picture controls, 3-52 to 3-53
  appearance, 3-51
  giving picture controls visual impact, 3-51
system attributes, 3-86 to 3-88
  list of attributes (table), 3-86
  reporting load failures, 3-87 to 3-88
  unsafe timer events, 3-86 to 3-87
timer controls, 3-64 to 3-65
autoscaling of graph plots, 3-82 to 3-83
axis label string functions
  ClearAxisItems, 4-46 to 4-47
  DeleteAxisItem, 4-55 to 4-56
  function tree, 4-5
  GetAxisItem, 4-82 to 4-83
  GetAxisItemLabelLength, 4-83 to 4-84
  GetNumAxisItems, 4-111
Index

InsertAxisItem, 4-128 to 4-129
ReplaceAxisItem, 4-213 to 4-214

axis scale functions
  function tree, 4-5
  GetAxisRange, 4-84 to 4-85
  GetAxisScalingMode, 4-86 to 4-87
  SetAxisRange, 4-224 to 4-226
  SetAxisScalingMode, 4-223 to 4-224

B

batch drawing, canvas controls, 3-54
binary switch controls
  definition, 1-12
  illustration, 1-12
bitmap, offscreen, 3-55
bitmap functions
  AllocBitmapData, 4-12 to 4-13
  DiscardBitmap, 4-61
  function tree, 4-8
  GetBitmapData, 4-87 to 4-89
  GetBitmapFromFile, 4-89 to 4-90
  GetBitmapInfo, 4-90 to 4-91
  GetCtrlBitmap, 4-92 to 4-93
  GetDisplayBitmap, 4-94 to 4-95
  GetPanelDisplayBitmap, 4-114 to 4-116
  NewBitmap, 4-151 to 4-153
  SetCtrlBitmap, 4-227 to 4-228
bitmap objects, 3-62 to 3-63
  creating, extracting, or discarding, 3-62
  displaying or copying, 3-63
  retrieving image data, 3-63
  Windows metafiles, 3-62 to 3-63
Bottom Edges option
  Alignment command, 2-21
  Distribution command, 2-21
Bring Panel to Front command, View menu, 2-19
bulletin board support, B-1

C

Callback Function field
  Edit Control dialog box, 2-11
  Edit Menu Bar dialog box, 2-9

Edit Panel dialog box, 2-10
callback functions
  assigning
    controls, 1-3
    User Interface Editor vs. programmatic method, 3-2
  associated with close controls
    (note), 2-25
  avoiding longjmp function (note), 3-7
diagram of callback function concept, 3-5
diagram of callback function
example program, 5-3
function reference
  function tree, 4-6 to 4-7
  InstallCtrlCallback, 4-133 to 4-134
  InstallMainCallback, 3-92 to 3-93, 4-135 to 4-136
  InstallMenuCallback, 4-136 to 4-137
  InstallMenuDimmerCallback, 4-137 to 4-138
  InstallPanelCallback, 4-138 to 4-139
  PostDeferredCall, 3-94, 4-189 to 4-190
generating code for
  All Callbacks command, 2-27
  Control Callbacks command, 2-28
  Main Function command, 2-26 to 2-27
  Menu Callbacks command, 2-28
  Panel Callbacks command, 2-28
precedence of callback functions, 3-91 to 3-92
processing events
  control events, 3-31
  menu bar events, 3-23
  panel events, 3-11
pseudocode example, 3-6 to 3-7
purpose and use, 1-2
responding to user interface events, 3-5 to 3-7
swallowing events, 3-92
timer callbacks, 3-64
using InstallMainCallback, 3-92 to 3-93
windows interrupt support functions
  function tree, 4-7
  GetCVITaskHandle, 4-98
GetCVIWindowHandle, 4-98 to 4-99
RegisterWinMsgCallback, 4-210 to 4-212
UnRegisterWinMsgCallback, 4-229

canvas controls, 3-53 to 3-59
attributes, 3-56 to 3-59
  ATTR_DRAW_POLICY values (table), 3-57
  ATTR_OVERLAPPED_POLICY values (table), 3-58
  ATTR_PEN_MODE values (table), 3-58
  list of attributes (table), 3-56 to 3-57
  pixel values for
    ATTR_PEN_PATTERN, 3-58 to 3-59
background color, 3-55
batch drawing, 3-54
canvas coordinate system, 3-54
clipping, 3-55
example programs
  canvas benchmark program, 5-6
drawing on canvas control, 5-6
  pie chart, 5-6
  using canvas control as drawing port for mouse, 5-6
functions for drawing on canvas, 3-53 to 3-54
  offscreen bitmap, 3-55
  pens, 3-55
  pixel values, 3-56
purpose and use, 1-21

canvas functions
  CanvasClear, 4-15 to 4-16
  CanvasDefaultPen, 4-16 to 4-17
  CanvasDimRect, 4-17 to 4-18
  CanvasDrawArc, 4-18 to 4-19
  CanvasDrawBitmap, 4-20 to 4-21
  CanvasDrawLine, 4-21 to 4-22
  CanvasDrawLineTo, 4-23
  CanvasDrawOval, 4-24 to 4-25
  CanvasDrawPoint, 4-25
  CanvasDrawPoly, 4-26 to 4-27
  CanvasDrawRectangle, 4-27 to 4-28
  CanvasDrawRoundedRectangle, 4-28 to 4-29
  CanvasDrawText, 4-30 to 4-31
  CanvasDrawTextAtPoint, 4-32 to 4-33
  CanvasEndBatchDrawing, 4-34
  CanvasGetClipRect, 4-35
  CanvasGetPenPosition, 4-35 to 4-36
  CanvasGetPixel, 4-36 to 4-37
  CanvasGetPixels, 4-37 to 4-39
  CanvasInvertRect, 4-39
  CanvasScroll, 4-40 to 4-41
  CanvasSetClipRect, 4-41 to 4-42
  CanvasSetPenPosition, 4-42
  CanvasStartBatchDraw, 4-43 to 4-44
  CanvasUpdate, 4-44 to 4-45
  function tree, 4-5 to 4-6

Case Sensitive option, Find UIR Objects dialog box, 2-18
Center Label command, Arrange menu, 2-22
Checked field, Edit Menu Bar dialog box, 2-9
CheckListItem function, 4-45 to 4-46
child panel
  example program, 5-4
  purpose and use, 3-11
ClearAxisItems function, 4-46 to 4-47
ClearListCtrl function, 4-47
ClearStripChart function, 4-47 to 4-48
clipboard functions
  ClipboardGetBitmap, 4-48 to 4-49
  ClipboardGetText, 4-49
  ClipboardPutBitmap, 4-49 to 4-50
  ClipboardPutText, 4-50
  example program, 5-6
  function tree, 4-8
clipping, canvas controls, 3-55
Close command, File menu, 2-4
code. See source files.
code generation, automatic. See CodeBuilder.

Code menu, User Interface Editor
  Generate submenu, 2-24 to 2-28
    All Callbacks command, 2-27
    All Code command, 2-25 to 2-26
    Control Callbacks command, 2-28
    enabled commands (note), 2-24
    Generate Code dialog box, 2-24
    illustration, 2-24
Index

Main Function command, 2-26 to 2-27
Menu Callbacks command, 2-28
Panel Callback command, 2-28
illustration, 2-23
Preferences command, 2-20 to 2-30
Always Append Code to End option, 2-30
Default Control Events option, 2-30
Default Panel Events option, 2-30
Set Target File command, 2-23 to 2-24
View command, 2-28 to 2-29
CodeBuilder
creating source code for GUI, 1-4 to 1-5
overview, 2-2 to 2-3
Color Preferences section, User Interface Editor Preferences dialog box, 2-32
Coloring tool, 2-2
colors
background color for canvas controls, 3-55
common color values for panel attributes (table), 3-17 to 3-18
example program changing image colors, 5-6
colors in list boxes, 5-5
functions accepting RGB color values, 4-10
generating with MakeColor function, 4-146 to 4-147
RGB color values for drawing three-dimensional objects, 4-79
command button controls
definition, 1-11
illustration, 1-11
operating, 1-11
commit events
control modes for generating, 1-3 to 1-4
hot, 1-4
indicator, 1-3
normal, 1-3
validate, 1-4
definition, 1-2
placed in GetUserEvent queue after being sent to callbacks, 3-92
ConfigurePrinter function, 4-51
confirm pop-up panel, 1-23
ConfirmPopup function, 4-51 to 4-52
Constant Name Assignment section, User Interface Editor Preferences dialog box, 2-32 to 2-33
Constant Name field
Edit Control dialog box, 2-11
Edit Menu Bar dialog box, 2-8 to 2-9
Edit Panel dialog box, 2-9
constant names, assigning
Assign Missing Constants command, Options menu, 2-33
constant name separator (_), 3-3
constant prefixes, 3-2 to 3-3
controls, 1-3, 3-3
menu bars, 3-3
menu items, 3-3
menus, 3-3
rules for User Interface Editor, 3-2 to 3-3
User Interface Editor method, 3-2 to 3-3
constant prefix, assigning
menus, 3-3
panels, 3-2 to 3-3
Control Appearance section, Edit Label/Value Pairs dialog box, 2-13
color attributes. See attributes.
Control Callbacks command, Generate menu, 2-28
Control command, Edit menu, 2-11 to 2-14. See also Edit Control dialog box.
Control Coordinates command, Arrange menu, 2-23
control functions. See also specific types of control functions.
function reference
DefaultCtrl, 4-54
DiscardCtrl, 4-61 to 4-62
DuplicateCtrl, 4-69 to 4-70
GetActiveCtrl, 4-80
GetCtrlAttribute, 4-91 to 4-92
GetCtrlBoundingRect, 4-93 to 4-94
GetCtrlVal, 4-96 to 4-97
NewCtrl, 4-153 to 4-154
SetActiveCtrl, 4-221
SetCtrlAttribute, 4-226 to 4-227
SetCtrlVal, 4-229 to 4-230
programming overview
control attributes. See attributes.
functions applicable to all controls,
3-28 to 3-29
list box and ring control functions,
3-29 to 3-30
processing control events, 3-31
to 3-32
text box functions, 3-30 to 3-31
Control Settings section, Edit Control dialog box, 2-12
Control Style command, Edit menu, 2-16
Control ZPlane Order command, Arrange menu, 2-22
controls, 1-7 to 1-21. See also control functions.
activating, 1-8
assigning callback functions, 1-3
assigning constant names, 1-3, 3-3
attributes. See attributes.
binary switch, 1-12
canvas. See canvas controls.
command button, 1-11
data types, 1-8
for ATTR_DATA_TYPE, 3-50
decorations, 1-16
definition, 1-7
data types, 1-8
graph. See graph controls.
LED, 1-12
list box, 1-14 to 1-16
numeric, 1-8 to 1-9
picture. See picture controls.
processing control events, 3-31 to 3-32
ring, 1-13 to 1-14
string, 1-9 to 1-10
strip chart. See strip chart controls.
text box, 1-10 to 1-11
text messages, 1-10
timer. See timer controls.
toggle button, 1-11 to 1-12
types of controls, 1-7 to 1-8
Copy command, Edit menu, 2-6
Copy Panel command, Edit menu, 2-6
Create menu, User Interface Editor
illustration, 2-16
Menu Bars command, 2-16
Panel command, 2-16
CreateMetaFont function, 4-52 to 4-53
cursor and mouse functions
function tree, 4-4
GetActiveGraphCursor, 4-80 to 4-81
GetCursorAttribute, 4-97
GetGlobalMouseState, 4-99 to 4-100
GetGraphCursor, 4-100 to 4-101
GetGraphCursorIndex, 4-101
GetMouseCursor, 4-110
GetRelativeMouseState, 4-118 to 4-119
GetWaitCursorState, 4-127
SetActiveGraphCursor, 4-221 to 4-222
SetCursorAttribute, 4-230 to 4-231
SetGraphCursor, 4-233
SetGraphCursorIndex, 4-234
SetMouseCursor, 4-241 to 4-242
SetWaitCursor, 4-248 to 4-249
cursors
cursor styles for
ATTR_CROSSHAIR_STYLE
(table), 3-78
data types, 1-8
elementary cursors (table), 1-17 to 1-18
free-form cursors (table), 1-17
to 1-18
snap-to-point cursors (table), 1-17
to 1-18
keyboard operation
cursor selection (table), 1-17 to 1-18
free-form cursors (table), 1-17
to 1-18
snap-to-point cursors (table), 1-17
to 1-18
styles for
ATTR_CURSOR_POINT_STYLE
and ATTR_TRACE_POINT_STYLE
(table), 3-78 to 3-79
values and cursor styles for
ATTR_MOUSE-CURSOR
(table), 3-19
customer communication, xxix, B-1 to B-2
Cut command, Edit menu, 2-6
Cut Panel command, Edit menu, 2-6
CVICALLBACK macro, 3-7
Index

D

data types
controls
valid data types, 1-8
values for ATTR_DATA_TYPE
attribute (table), 3-50
plot arrays, 4-11 to 4-12
decorations, 1-16
Default Control Events option, Preferences
command, 2-30
Default Panel Events option, Preferences
command, 2-30
DefaultCtrl function, 4-54
DefaultPanel function, 4-54 to 4-55
Delete command, Edit menu, 2-6
DeleteAxisItem function, 4-44 to 4-56
DeleteGraphPlot function, 4-56 to 4-57
DeleteImage function, 4-58
DeleteListItem function, 4-58 to 4-59
DeleteTextBoxLine function, 4-59
developing graphical user interfaces (GUI).
See graphical user interface (GUI),
building.
Dimmed field, Edit Menu Bar dialog
box, 2-9
DirSelectPopup function, 4-60
DiscardBitmap function, 4-61
DiscardCtrl function, 4-61 to 4-62
DiscardMenu function, 4-62
DiscardMenuBar function, 4-63
DiscardMenuItem function, 4-63
DiscardPanel function, 4-64
DiscardSubMenu function, 4-64
DisplayImageFile function, 4-65
DisplayPanel function, 4-66 to 4-67
DisplayPCXFile function, 4-66
Distribute Vertical Centers command,
Arrange menu, 2-22
Distribution command, Arrange menu, 2-21
to 2-22
dltd
control attributes
ATTR_CHECK_RANGE values
(table), 3-51
documentation
conventions used in manual, xx
organization of manual, xix-xx
DOSColorToRGB function, 4-67 to 4-68
DOSCompatWindow function, 4-68
drawing. See canvas controls; canvas
functions.
DuplicateCtrl function, 4-69 to 4-70
DuplicatePanel function, 4-70 to 4-71

E

Edit button, Find UIR Objects dialog
box, 2-18
Edit Control dialog box
Control Settings section, 2-12
Edit Label/Value Pairs dialog box, 2-12
to 2-13
Quick Edit Window, 2-14
Source Code Connection, 2-11
Edit Label/Value Pairs dialog box
Control Appearance section, 2-13
illustration, 2-12
Label Appearance section, 2-13
purpose and use, 2-13
Edit menu, User Interface Editor
Apply Default Font command, 2-15
Control command, 2-11 to 2-14
Control Style command, 2-16
Copy command, 2-6
Copy Panel command, 2-6
Cut command, 2-6
Cut Panel command, 2-6
Delete command, 2-6
illustration, 2-5
Menu Bars command, 2-7 to 2-9
Panel command, 2-9 to 2-11
Paste command, 2-6
Redo command, 2-5
Set Default Font command, 2-15
Tab Order command, 2-14 to 2-15
Undo command, 2-5
when commands are enabled (note), 2-5
Edit Menu Bar dialog box
available options, 2-8 to 2-9
illustration, 2-8
Edit Panel dialog box
  Panel Attributes section, 2-10
  Quick Edit Window section, 2-10
to 2-11
  Source Code Connection section, 2-9
Edit Tab Order dialog box, 2-15
Editing tool, 2-1
electronic technical support, B-1 to B-2
e-mail support, B-2
EmptyMenu function, 4-71
EmptyMenuBar function, 4-71 to 4-72
error conditions, User Interface Library, A-1
to A-6
error reporting, User Interface Library, 4-12
event functions
  GetUserEvent, 3-8 to 3-9, 4-125
  ProcessDrawEvents, 3-93, 4-195
  ProcessSystemEvents, 3-93 to 3-94,
  4-195 to 4-196
  QueueUserEvent, 4-197
  SetIdleEventRate, 3-92 to 3-93, 4-235
event loops
  definition, 1-2
  example program, 5-4 to 5-5
  GetUserEvent function for responding to
events, 3-8 to 3-9
  processing events
    control events, 3-31 to 3-32
    menu bar events, 3-22 to 3-23
  precedence of callback functions, 3-91
to 3-92
  swallowing events, 3-92
  unsafe timer events, 3-86 to 3-87
  example programs, 5-1 to 5-6
Exit LabWindows/CVI command, File menu, 2-4

F

FakeKeystroke function, 3-94, 4-72
fax and telephone technical support, B-2
faxback support, B-1
File menu, User Interface Editor
  Add File to Project command, 2-4
  Close command, 2-4
  Exit LabWindows/CVI command, 2-4
  illustration, 2-3
  New command, 2-4
  Open command, 2-4
  Print command, 2-4
  Read Only command, 2-4
  Save command, 2-4
  Save All command, 2-4
  Save As command, 2-4
  Save Copy As command, 2-4
file select pop-up panel, 1-23 to 1-24
FileSelectPopup function, 4-72 to 4-74
Find button, Find UIR Objects dialog box, 2-18
Find Next button, Find UIR Objects dialog box, 2-19
Find Prev button, Find UIR Objects dialog box, 2-18
Find UIR Objects command, View menu, 2-17 to 2-18
Find UIR Objects dialog box
  Case Sensitive option, 2-18
  Edit button, 2-18
  Find button, 2-18
  Find Next button, 2-18
  Find Prev button, 2-18
  illustration, 2-17
  Regular Expression option, 2-18
  search criteria in Search By ring control, 2-17 to 2-18
  Stop button, 2-18
  Wrap option, 2-18
fonts
  CreateMetaFont function, 4-52 to 4-53
  FontSelectPopup function, 4-74 to 4-77
  functions using fonts as input parameters, 4-10
  metafonts
    included with
      LabWindows/CVI, 1-25
typefaces native to each platform, 1-25
panel attributes
  host fonts, 3-21
  metafonts included with
    LabWindows/CVI, 3-21
platform independent fonts on PCs and UNIX, 3-20
platform independent metafonts on PCs and UNIX, 3-20 to 3-21
user defined metafonts, 3-21
valid font values (table), 3-20
setting and applying default fonts, 2-15
typefaces native to each platform, 1-25
FontSelectPopup function, 4-74 to 4-77
FTP support, B-2

G
Generate All Code dialog box, 2-25
Generate Code dialog box, 2-24
Generate Main Function dialog box, 2-26
Generate menu, 2-24 to 2-28
  All Callbacks command, 2-27
  All Code command, 2-25 to 2-26
  Control Callbacks command, 2-28
  enabled commands (note), 2-24
  Generate Code dialog box, 2-24
  illustration, 2-24
  Main Function command, 2-26 to 2-27
  Menu Callbacks command, 2-28
  Panel Callback command, 2-28
generating code automatically. See CodeBuilder.
generic message pop-up panel, 1-22
GenericMessagePopup function, 4-77 to 4-79
generic message pop-up panel, 1-22
GenericMessagePopup function, 4-77 to 4-79
gerometric panel attributes, 3-18
Get3dBorderColors function, 4-79 to 4-80
GetActiveCtrl function, 4-80
GetActiveGraphCursor function, 4-80 to 4-81
GetActivePanel function, 4-81
GetAxisItem function, 4-82 to 4-83
GetAxisItemLabelLength function, 4-83 to 4-84
GetAxisRange function, 4-84 to 4-85
GetAxisScalingMode function, 4-86 to 4-87
GetBimapData function, 4-87 to 4-89
GetBitmapFromFile function, 4-89 to 4-90
GetBitmapInfo function, 4-90 to 4-91
GetCtrlAttribute function, 4-91 to 4-92
GetCtrlBitmap function, 4-92 to 4-93
GetCtrlBoundingRect function, 4-93 to 4-94
GetCtrlDisplayBitmap function, 4-94 to 4-95
GetCtrlIndex function, 4-95 to 4-96
GetCtrlVal function, 4-96 to 4-97
GetCursorAttribute function, 4-97
GetCVITaskHandle function, 4-98
GetCVIWindowHandle function, 4-98 to 4-99
Index

GetGlobalMouseState function, 4-99 to 4-100
GetGraphCursor function, 4-100 to 4-101
GetGraphCursorIndex function, 4-101
GetImageBits function, 4-102 to 4-104
GetImageInfo function, 4-105 to 4-106
GetIndexFromValue function, 4-106 to 4-107
GetLabelFromIndex function, 4-107
GetLabelLengthFromIndex function, 4-108
GetPosition function, 4-108 to 4-109
GetMenuBarAttribute function, 4-109 to 4-110
GetMouseCursor function, 4-110
GetNumAxisItems function, 4-111
GetNumCheckedItems function, 4-112
GetNumListItems function, 4-112 to 4-113
to 4-114
GetPanelAttribute function, 4-114
to 4-116
GetPanelDisplayBitmap function, 4-114 to 4-116
GetPanelMenuBar function, 4-116
GetPlotAttribute function, 4-116 to 4-117
GetPrintAttribute function, 4-117 to 4-118
GetRelativeMouseState function, 4-118 to 4-119
GetScreenSize function, 4-119
GetSharedMenuBarEventPanel function, 4-119 to 4-120
GetSleepPolicy function, 4-120 to 4-121
GetSystemAttribute function, 4-120 to 4-121
GetSystemPopupsAttribute function, 4-121
GetTextBoxLine function, 4-121 to 4-122
GetTextBoxLineLength function, 4-122 to 4-123
GetTextDisplaySize function, 4-123
GetTraceAttribute function, 4-123 to 4-124
GetUILErrorString function, 4-124
to 4-125
GetUserEvent function. See also event loops.
  description, 4-125
  purpose, 3-92
  responding to events, 3-8 to 3-9
  diagram of event loop concept, 3-8
  pseudocode example, 3-8 to 3-9
GetValueFromIndex function, 4-126
GetValueLengthFromIndex function, 4-126 to 4-127
GetWaitCursorState function, 4-127
GOT_FOCUS event, 1-2
graph control functions
  AllocImageBits, 4-14 to 4-15
  ClearStripChart, 4-47 to 4-48
  DeleteGraphPlot function, 4-56 to 4-57
  DeleteImage, 4-58
  DisplayImageFile, 4-65
  DisplayPCXFile, 4-66
  function tree, 4-4
  GetActiveGraphCursor, 4-80 to 4-81
  GetAxisRange, 4-84 to 4-85
  GetCursorAttribute, 4-97
  GetGraphCursor, 4-100 to 4-101
  GetGraphCursorIndex, 4-101
  GetImageBits, 4-102 to 4-104
  GetImageInfo, 4-105 to 4-106
  GetPlotAttribute, 4-116 to 4-117
  GetTraceAttribute, 4-123 to 4-124
  graph and strip chart controls, 3-66 to 3-67
  PlotArc, 4-160 to 4-162
  PlotBitMap, 4-162 to 4-163
  PlotIntensity, 4-163 to 4-166
  PlotLine, 4-166 to 4-167
  PlotOval, 4-167 to 4-168
  PlotPoint, 4-169
  PlotPolygon, 4-170 to 4-171
  PlotRectangle, 4-171 to 4-172
  PlotScaledIntensity, 4-173 to 4-176
  PlotStripChart, 4-176 to 4-178
  PlotStripChartPoint, 4-178 to 4-179
  PlotText, 4-179 to 4-180
  PlotWaveform, 4-181 to 4-182
  PlotX, 4-183 to 4-184
  PlotXY, 4-184 to 4-185
  PlotY, 4-185 to 4-187
  RefreshGraph, 4-210
  ResetTimer, 4-216 to 4-217
  ResumeTimerCallbacks, 4-217
  SetActiveGraphCursor, 4-221 to 4-222
  SetAxisRange, 4-224 to 4-226
SetCursorAttribute, 4-230 to 4-231
SetGraphCursor, 4-233
SetGraphCursorIndex, 4-234
SetImageBits, 4-236 to 4-238
SetPlotAttribute, 4-244 to 4-245
SetTraceAttribute, 4-247 to 4-248
SuspendTimerCallbacks, 4-249

graph controls. See also graph control functions.
attributes
cursor styles for
  ATTR_CROSSHAIR_STYLE
    (table), 3-78
discussion of specific attributes, 3-77
to 3-80
line styles for ATTR_LINE_STYLE
  (table), 3-80
list of attributes (table), 3-68 to 3-77
plot origin, 3-80 to 3-81
plot styles for ATTR_PLOT_STYLE
  (table), 3-80
styles for
  ATTR_CURSOR_POINT_STYLE
  and
  ATTR_TRACE_POINT_STYLE
    (table), 3-78 to 3-79
two Y axes, 3-81
values for ATTR_PLOT_ORIGIN
  (table), 3-81
definition, 1-16
two Y axes, 3-81
definition, 1-16
example program, 5-3
illustration, 1-16
operating, 1-16 to 1-19
optimizing, 3-82 to 3-85
controlling refresh, 3-83 to 3-84
example canvas benchmark
program, 5-6
memory usage, 3-84 to 3-85
speed, 3-82 to 3-84
processing events, 3-67 to 3-68
zooming and panning on graphs, 1-19
graph cursor functions. See cursor and mouse functions.
graph cursors. See cursors.
graph plotting and deleting functions
  DeleteGraphPlot function, 4-56 to 4-57
GetPlotAttribute, 4-116 to 4-117
PlotArc, 4-160 to 4-162
PlotBitMap, 4-162 to 4-163
PlotIntensity, 4-163 to 4-166
PlotLine, 4-166 to 4-167
PlotOval, 4-167 to 4-168
PlotPoint, 4-169
PlotPolygon, 4-170 to 4-171
PlotRectangle, 4-171 to 4-172
PlotText, 4-179 to 4-180
PlotWaveform, 4-181 to 4-182
PlotX, 4-183 to 4-184
PlotXY, 4-184 to 4-185
PlotY, 4-185 to 4-187
RefreshGraph, 4-210
SetPlotAttribute, 4-244 to 4-245
graph pop-up panel, 1-24
graphical user interface (GUI)
  building. See graphical user interface
    (GUI), building.
  components, 1-1
  decorations, 1-16
  fonts, 1-25
  illustration, 1-1
  pop-up panels, 1-21 to 1-24
graphical user interface (GUI), building. See
  also User Interface Editor.
  assigning constant names in User
    Interface Editor, 3-2 to 3-3
  bitmap objects, 3-62 to 3-63
    creating, extracting, or
discarding, 3-62
  displaying or copying, 3-63
  retrieving image data, 3-63
  Windows metafiles, 3-62 to 3-63
canvas controls, 3-53 to 3-59
  attributes, 3-56 to 3-59
  background color, 3-55
  batch drawing, 3-54
canvas coordinate system, 3-54
canvas coordinate system, 3-54
  clipping, 3-55
canvas controls, 3-53 to 3-59
  functions for drawing on canvas,
    3-53 to 3-54
  offscreen bitmap, 3-55
  pens, 3-55
  pixel values, 3-56
control functions
  for all controls, 3-28 to 3-29
  list boxes and rings, 3-29 to 3-30
text boxes, 3-30 to 3-31
controlling the interface
  basic methods, 1-2
  callback functions, 3-5 to 3-7
  GetUserEvent function (event loop), 3-8 to 3-9
  User Interface Editor vs. programmatic method, 3-2
user interface events, 3-3 to 3-9
controls
  attributes, 3-32 to 3-33
  callback functions, 3-31
  event loops, 3-31 to 3-32
  processing events, 3-31 to 3-32
development procedure, 3-1
events, 1-2
  callback functions for responding to events, 3-5 to 3-7
  control modes for generating commit events, 1-3 to 1-4
  event types and information passed to program (table), 3-4
  GetUserEvent function (event loop) for responding to events, 3-8 to 3-9
example program, 5-4
general guidelines, 3-1
generating hard copy output
  functions, 3-88
  hard copy attributes, 3-89 to 3-91
graph and strip chart controls
  attributes, 3-68 to 3-80
  functions, 3-65 to 3-67
  processing events, 3-67 to 3-68
menu bars, 1-6 to 1-7
  attributes, 3-24 to 3-25
  functions, 3-21 to 3-22
  processing events, 3-22 to 3-23
panels
  attributes (table), 3-12 to 3-16
  functions, 3-9 to 3-10
  processing events, 3-11
picture controls, 3-51 to 3-53
  attributes, 3-52 to 3-53
image formats (table), 3-52
  simple picture control, 3-52
pop-up panels, 3-85 to 3-86
recompiling program after saving .uir file, 3-1
rect and point structures, 3-59 to 3-61
  comparing or obtaining values, 3-61
  functions and macros for creating, 3-60
  modifying, 3-60 to 3-61
source code connection, 1-3
special functions
  FakeKeystroke, 3-94
  InstallMainCallback, 3-92 to 3-93
  PostDeferredCall, 3-94
  precedence of callback functions, 3-91 to 3-92
  ProcessSystemEvents, 3-93 to 3-94, 4-195 to 4-196
  QueueUserEvent, 3-94
  QuitUserInterface, 3-94
  RunUserInterface, 3-91
  SetIdleEventRate, 3-92 to 3-93
  swallowing events, 3-92
system attributes, 3-86 to 3-88
  list of attributes (table), 3-86
  reporting load failures, 3-87 to 3-88
  unsafe timer events, 3-86 to 3-87
timer controls, 3-63 to 3-65
  attributes, 3-64 to 3-65
  operations, 3-65
timer callbacks, 3-64
timer control functions, 3-63 to 3-65
GUI. See graphical user interface (GUI).

H

hard copy. See also printing functions.
  attributes (table), 3-89
  compatible printers, 4-9
  configuring devices, 4-9 to 4-10
discussion of specific attributes, 3-90 to 3-91
  functions for generating, 3-88
  limitations with UNIX (note), 3-90
obtaining output, 4-9
values for ATTR_COLOR_MODE
(table), 3-91
header files
created by saving .uir file in User
Interface Editor, 1-3
previewing, 2-19
HidePanel function, 4-128
hierarchy buttons, Edit Menu Bar dialog
box, 2-9
Horizontal Centers option
  Alignment command, 2-20
  Distribution command, 2-22
Horizontal Compress option, Distribution
command, 2-22
Horizontal Gap option, Distribution
command, 2-22
host fonts, 3-21
hot control mode for commit events
definition, 1-4
rules for generation of commit
events, 1-4
hot keys. See shortcut keys.

I
IDs for interface objects. See constant
names, assigning.
images. See picture control functions;
picture controls.
immediate action menus, 1-6
include files
  include file for User Interface Library
(userint.h), 4-12
required for referencing resource IDs
and callback functions, 3-2
updating by recompiling programs, 3-1
indicator control mode for commit
events, 1-3
Insert New Item field, Edit Menu Bar dialog
box, 2-9
Insert Separator field, Edit Menu Bar dialog
box, 2-9
InsertAxisItem function, 4-128 to 4-129
InsertListItem function, 4-130 to 4-131
InsertSeparator function, 4-132
InsertTextBoxLine function, 4-132 to 4-133
InstallCtrlCallback function, 4-133 to 4-134
InstallMainCallback function, 3-92 to 3-93,
  4-135 to 4-136
InstallMenuCallback function, 4-136
to 4-137
InstallMenuDimmerCallback function,
  4-137 to 4-138
InstallPanelCallback function, 4-138
to 4-139
InstallPopup function, 4-139
intensity plot sample program, 5-5
IsListItemChecked function, 4-140
Item field, Edit Menu Bar dialog box, 2-8

K
key modifiers. See modifier key.
keyboard input, example program, 5-5

L
Label Appearance section, Edit Label/Value
Pairs dialog box, 2-13
Labeling tool, 2-2
Label/Value Pairs button, 2-12. See also
  Edit Label/Value Pairs dialog box.
LabWindows for DOS compatibility
  functions. See LW DOS compatibility
  functions.
LED controls, 1-12
Left Edges option
  Alignment command, 2-20
  Distribution command, 2-22
LEFT_CLICK event, 1-2
Library menu, User Interface Editor, 2-30
line styles for ATTR_LINE_STYLE
  (table), 3-79
list box control functions
  CheckListItem, 4-45 to 4-46
  ClearListCtrl, 4-47
  DeleteListItem, 4-58 to 4-59
  function tree, 4-3
  GetCtrlIndex, 4-95 to 4-96
GetIndexFromValue, 4-106 to 4-107
GetLabelFromIndex, 4-107
GetLabelLengthFromIndex, 4-108
GetListItemImage, 4-108 to 4-109
GetNumCheckedItems, 4-112
GetNumListItems, 4-112 to 4-113
GetValueFromIndex, 4-126
GetValueLengthFromIndex, 4-126 to 4-127
InsertListItem, 4-130 to 4-131
IsListItemChecked, 4-140
list of functions, 3-29 to 3-30
ReplaceListItem, 4-214 to 4-215
SetCtrlIndex, 4-229
SetListItemImage, 4-240
copylist box controls. See also Edit Label/Value Pairs dialog box.
definition, 1-14
equivalent program, 5-4, 5-5
description, 1-14
operating, 1-15 to 1-16
load failures, reporting, 3-87 to 3-88
Load From Text Format command, Options menu, 2-34
LoadMenuBar function, 4-140 to 4-141
LoadMenuBarEx function, 4-142 to 4-143
LoadPanel function, 4-143 to 4-145
LoadPanelEx function, 4-145 to 4-146
longjmp function, avoiding in callback functions (note), 3-7
LW DOS compatibility functions
ConfigurePrinter, 4-51
DisplayPCXFile, 4-66
DOSColorToRGB, 4-67 to 4-68
DOSCompatWindow, 4-68
function tree, 4-9
GetUILLErrorString, 4-124

M

Main Function command, Generate menu, 2-26 to 2-27
MakeColor function, 4-146 to 4-147
MakePoint function, 4-148
MakeRect function, 4-148 to 4-149
manual. See documentation.
memory usage, optimizing, 3-84 to 3-85
Menu Bar Constant Prefix field, Edit Menu Bar dialog box, 2-8
menu bar functions. See also menu functions; menu item functions.
DiscardMenuBar, 4-63
EmptyMenuBar, 4-71 to 4-72
function tree, 4-2
GetMenuBarAttribute, 4-109 to 4-110
GetPanelMenuBar, 4-116
GetSharedMenuBarEventPanel, 4-119 to 4-120
LoadMenuBar, 4-140 to 4-141
LoadMenuBarEx, 4-142 to 4-143
NewMenuBar, 4-155
programming graphical user interface (GUI), 3-21 to 3-22
SetMenuBarAttribute, 4-241 to 4-242
SetPanelMenuBar, 4-243
menu bars, 1-6 to 1-7
assigning constant names
constant prefix for menus, 3-3
menu items, 3-3
requirements, 3-3
submenus (note), 3-3
attributes
cross mask for masking three bit fields (table), 3-28
menu and menu item attributes (table), 3-24 to 3-25
modifiers and virtual keys for shortcut keys (table), 3-26 to 3-27
definition, 1-6
illustration, 1-6
immediate action menus, 1-6
operating, 1-6 to 1-7
processing menu bar events, 3-22 to 3-23
pull-down menus, 1-6
Menu Bars command
Create menu, 2-16
Edit menu, 2-7 to 2-9
Menu Bars List dialog box. See also Edit Menu Bar dialog box.
available command buttons, 2-7

© National Instruments Corporation
illustration, 2-7
Menu Callbacks command, Generate menu, 2-28
menu functions. See also menu bar functions; menu item functions.
  DiscardMenu, 4-62
  DiscardSubMenu, 4-64
  EmptyMenu, 4-71
  example program, 5-3
  NewMenu, 4-154 to 4-155
  NewSubMenu, 4-160
  RunPopupMenu, 4-218 to 4-219
menu item functions. See also menu bar functions; menu functions.
  DiscardMenuItem, 4-63
  InsertSeparator, 4-132
  NewMenuItem, 4-156 to 4-158
message pop-up panel, 1-22
MessagePopup function, 4-149
metafonts
  included with
    LabWindows/CVI, 1-25, 3-21
  platform independent fonts on PCs and UNIX, 3-20
  typefaces native to each platform, 1-25
  user defined metafonts, 3-21
miscellaneous functions
  CreateMetaFont, 4-52 to 4-53
  Get3dBorderColors, 4-79 to 4-80
  GetMouseCursor, 4-110
  GetScreenSize, 4-119
  GetSystemAttribute, 4-120 to 4-121
  GetTextDisplaySize, 4-123
  GetWaitCursorState, 4-127
  MakeColor, 4-146 to 4-147
  SetMouseCursor, 4-241 to 4-242
  SetSystemAttribute, 4-226
  SetWaitCursor, 4-248 to 4-249
modifier key
  attribute values, 3-26, 4-157
  representation in source code, 3-26
  Modifier Key field, Edit Menu Bar dialog box, 2-9
mouse functions. See cursor and mouse functions.
  mouse operation of cursors, 1-18
  mouse state, example program, 5-5
Move Backward option, Control ZPlane Order command, 2-22
Move Forward option, Control ZPlane Order command, 2-22
Move to Back option, Control ZPlane Order command, 2-22
Move to Front option, Control ZPlane Order command, 2-22
MultiFileSelectPopup function, 4-150 to 4-151
N
  New command, File menu, 2-4
  NewBitmap function, 4-151 to 4-153
  NewCtrl function, 4-153 to 4-154
  NewMenu function, 4-154 to 4-155
  NewMenuBar function, 4-155
  NewMenuItem function, 4-156 to 4-158
  NewPanel function, 4-159 to 4-160
  NewSubMenu function, 4-160
  Next Panel command, View menu, 2-19
  Next Tool command, Options menu, 2-31
numeric controls
  definition, 1-8
  illustration, 1-8
  operating, 1-8 to 1-9
numeric formats for ATTR_FORMAT, 3-51
O
  offscreen bitmap, 3-55
Open command, File menu, 2-4
Operate Visible Panels command, Options menu, 2-31
Operating tool, 2-2
optimizing graph controls, 3-82 to 3-85
  controlling refresh, 3-83 to 3-84
  example canvas benchmark program, 5-6
  memory usage, 3-84 to 3-85
  speed, 3-82 to 3-84
Options menu
Project window
  Track Include File Dependencies command, 3-1
User Interface Editor
  Assign Missing Constants command, 2-33
  illustration, 2-31
  Load From Text Format command, 2-34
  Next Tool command, 2-31
  Operate Visible Panels command, 2-31
  Preferences command, 2-32 to 2-33
  Save in Text Format command, 2-33

panels
  assigning constant prefix, 3-2 to 3-3
  attributes, 3-12 to 3-22
  color values (table), 3-17 to 3-18
  fonts, 3-20 to 3-21
  frame style values, 3-18
  geometric attributes (table), 3-18
  list of attributes (table), 3-12 to 3-16
  programming considerations, 3-17 to 3-20
  values and cursor styles for ATTR_MOUSE-CURSOR (table), 3-19
  child panel role, 3-11
  copying or cutting, 2-6
  definition, 1-5
  Edit Panel dialog box, 2-9 to 2-11
  illustration, 1-5
  operating, 1-5 to 1-6
  processing panel events, 3-11
  Show/Hide Panels command, View menu, 2-19
  panning on graphs
    definition, 1-19
    enabling panning, 1-19
  Paste command, Edit menu, 2-6
  pen functions
    CanvasDefaultPen, 4-16 to 4-17
    CanvasGetPenPosition, 4-35 to 4-36
    CanvasSetPenPosition, 4-42
  pens for canvas controls, 3-55
  picture control functions
    AllocImageBits, 4-14 to 4-15
    DeleteImage, 4-58
    DisplayImageFile, 4-65
    DisplayPCXFile, 4-66
    function tree, 4-5
    GetImageBits, 4-102 to 4-104
    GetImageInfo, 4-105 to 4-106
    SetImageBits, 4-236 to 4-238
  picture controls, 3-51 to 3-53
  attributes, 3-52 to 3-53
    appearance, 3-51
    giving picture controls visual impact, 3-51
    definition, 1-20

Panel Attributes section, Edit Panel dialog box, 2-10
Panel Callback command, Generate menu, 2-28
Panel command
  Create menu, 2-16
  Edit menu, 2-9 to 2-11
  panel functions. See also pop-up panel functions.
    DefaultPanel, 4-54 to 4-55
    DiscardPanel, 4-64
    DisplayPanel, 4-66 to 4-67
    DuplicatePanel, 4-70 to 4-71
    function tree, 4-2
    GetActivePanel, 4-81
    GetPanelAttribute, 4-114
    HidePanel, 4-128
    LoadPanel, 4-143 to 4-145
    LoadPanelEx, 4-145 to 4-146
    NewPanel, 4-159 to 4-160
    programming graphical user interface (GUI), 3-9 to 3-10
    RecallPanelState, 4-198 to 4-199
    SavePanelState, 4-220
    SetActivePanel, 4-222
    SetPanelAttribute, 4-242
    SetPanelPos, 4-243 to 4-244
    ValidatePanel, 4-250

Panels
  assigning constant prefix, 3-2 to 3-3
  attributes, 3-12 to 3-22
  color values (table), 3-17 to 3-18
  fonts, 3-20 to 3-21
  frame style values, 3-18
  geometric attributes (table), 3-18
  list of attributes (table), 3-12 to 3-16
  programming considerations, 3-17 to 3-20
  values and cursor styles for ATTR_MOUSE-CURSOR (table), 3-19
  child panel role, 3-11
  copying or cutting, 2-6
  definition, 1-5
  Edit Panel dialog box, 2-9 to 2-11
  illustration, 1-5
  operating, 1-5 to 1-6
  processing panel events, 3-11
  Show/Hide Panels command, View menu, 2-19
  panning on graphs
    definition, 1-19
    enabling panning, 1-19
  Paste command, Edit menu, 2-6
  pen functions
    CanvasDefaultPen, 4-16 to 4-17
    CanvasGetPenPosition, 4-35 to 4-36
    CanvasSetPenPosition, 4-42
  pens for canvas controls, 3-55
  picture control functions
    AllocImageBits, 4-14 to 4-15
    DeleteImage, 4-58
    DisplayImageFile, 4-65
    DisplayPCXFile, 4-66
    function tree, 4-5
    GetImageBits, 4-102 to 4-104
    GetImageInfo, 4-105 to 4-106
    SetImageBits, 4-236 to 4-238
  picture controls, 3-51 to 3-53
  attributes, 3-52 to 3-53
    appearance, 3-51
    giving picture controls visual impact, 3-51
    definition, 1-20
example program, 5-4
image formats (table), 1-20, 3-52
simple picture control, 3-52
pixel functions
  CanvasGetPixel, 4-36 to 4-37
  CanvasGetPixels, 4-37 to 4-39
pixel values, canvas controls, 3-56
plot array data types, 4-11 to 4-12
plot control functions. See graph control functions.
plot origin, graph and strip chart controls, 3-80 to 3-81
plot styles for ATTR_PLOT_STYLE (table), 3-80
PlotArc function, 4-160 to 4-162
PlotBitMap function, 4-162 to 4-163
PlotIntensity function, 4-163 to 4-166
PlotLine function, 4-166 to 4-167
PlotOval function, 4-167 to 4-168
PlotPoint function, 4-169
PlotPolygon function, 4-170 to 4-171
PlotRectangle function, 4-171 to 4-172
PlotScaledIntensity function, 4-173 to 4-176
PlotStripChart function, 4-176 to 4-178
PlotStripChartPoint function, 4-178 to 4-179
PlotText function, 4-179 to 4-180
PlotWaveform function, 4-181 to 4-182
PlotX function, 4-183 to 4-184
PlotXY function, 4-184 to 4-185
PlotY function, 4-185 to 4-187
point functions. See rect and point functions.
point structures. See rect and point structures.
PointEqual function, 4-187
PointPinnedToRect function, 4-188
PointSet function, 4-188 to 4-189
pop-up menus, User Interface Editor window, 2-2
pop-up panel functions
  ConfirmPopup, 4-51 to 4-52
  DirSelectPopup, 4-60
  FileSelectPopup function, 4-72 to 4-74
  FontSelectPopup, 4-74 to 4-77
  function tree, 4-6
  GenericMessagePopup, 4-77 to 4-79
  GetSystemPopupsAttribute, 4-121
  InstallPopup, 4-139
  MessagePopup, 4-149
  MultiFileSelectPopup, 4-150 to 4-151
  PromptPopup, 4-196
  RemovePopup, 4-212
  SetFontPopupDefaults, 4-231 to 4-233
  SetSystemPopupsAttribute, 4-247
  WaveformGraphPopup, 4-251
  XGraphPopup, 4-252
  XYGraphPopup, 4-252 to 4-253
  YGraphPopup, 4-253
pop-up panels, 1-21 to 1-24
  confirm pop-up panels, 1-23
definition, 1-21
element program, 5-3
file select pop-up panel, 1-23 to 1-24
functions for accessing predefined pop-up panels, 3-85 to 3-86
generic message pop-up panel, 1-22
  graph pop-up panel, 1-24
message pop-up panel, 1-22
programming overview, 3-85 to 3-86
prompt pop-up panel, 1-22
PostDeferredCall function, 4-189 to 4-190
precedence of callback functions, 3-91 to 3-92
Preferences command
  Code menu, 2-29 to 2-30
    Always Append Code to End option, 2-30
    Default Control Events option, 2-30
    Default Panel Events option, 2-30
    Preferences menu (illustration), 2-29
  Options menu, 2-31 to 2-34. See also User Interface Editor Preferences dialog box.
Preview User Interface Header File command, View menu, 2-19
Previous Panel command, View menu, 2-19
Print command, File menu, 2-4
PrintCtrl function, 4-190 to 4-191
printing functions
  GetPrintAttribute, 4-117 to 4-118
  PrintCtrl, 4-190 to 4-191
  PrintPanel, 4-191 to 4-193
  PrintTextBuffer, 4-193 to 4-194
Index

PrintTextFile, 4-194 to 4-195
SetPrintAttribute, 4-245
printing hardcopy. See hard copy.
PrintPanel function, 4-191 to 4-193
PrintTextBuffer function, 4-193 to 4-194
PrintTextFile function, 4-194 to 4-195
ProcessDrawEvents function, 3-93, 4-195
ProcessSystemEvents function, 3-93 to 3-94, 4-195 to 4-196
program shell, building. See CodeBuilder.
programming examples, 5-1 to 5-6
programming graphical user interfaces. See graphical user interface (GUI), building.
prompt pop-up panel, 1-22
PromptPopup function, 4-196
pull-down menus, 1-6
push button controls. See command button controls.

Q
QueueUserEvent function, 4-197
Quick Edit Window
   Edit Control dialog box, 2-14
   Edit Panel dialog box, 2-10 to 2-11
QuitUserInterface function
   definition, 3-94
   description, 4-197 to 4-198

R
Read Only command, File menu, 2-4
RecallPanelState function, 4-198 to 4-199
rect and point functions
   function tree, 4-7 to 4-8
   MakePoint, 4-148
   MakeRect, 4-148 to 4-149
   PointEqual, 4-187
   PointPinnedToRect, 4-188
   PointSet, 4-188 to 4-189
   RectBottom, 4-199
   RectCenter, 4-199 to 4-200
   RectContainsRect, 4-200
   RectEmpty, 4-201 to 4-203
   RectEqual, 4-202
   RectGrow, 4-202 to 4-203
   RectIntersection, 4-203 to 4-204
   RectMove, 4-204
   RectOffset, 4-204 to 4-205
   RectRight, 4-205
   RectSameSize, 4-206
   RectSet, 4-206 to 4-207
   RectSetBottom, 4-207
   RectSetCenter, 4-207 to 4-208
   RectSetFromPoints, 4-208
   RectSetRight, 4-208
   RectUnion, 4-209
rect and point structures, 3-59 to 3-61
   comparing or obtaining values, 3-61
   functions and macros for creating, 3-60
   modifying, 3-60 to 3-61
   purpose and use, 3-59
Redo command, Edit menu, 2-5
refresh rate for graphs, 3-83 to 3-84
RefreshGraph function, 4-210
RegisterWinMsgCallback function, 4-210 to 4-212
Regular Expression option, Find UIR Objects dialog box, 2-18
RemovePopup function, 4-212
ReplaceAxisItem function, 4-213 to 4-214
ReplaceListItem function, 4-214 to 4-215
ReplaceTextBoxLine function, 4-215
reporting load failures, 3-87 to 3-88
ResetTextBox function, 4-216
ResetTimer function, 4-216 to 4-217
resource ID
   controls, 2-11
   menu bar, 2-8
   panels, 2-9
   ResumeTimerCallbacks function, 4-217
RGB colors. See colors.
Right Edges option
   Alignment command, 2-20
   Distribution command, 2-22
ring controls. See also Edit Label/Value Pairs dialog box.
   control functions, 3-29 to 3-30
   definition, 1-13
   operating, 1-13 to 1-14
   pop-up format (illustration), 1-13
Run menu, User Interface Editor, 2-30
RunPopupMenu function, 4-218 to 4-219
RunUserInterface function
  description, 4-219 to 4-220
  purpose, 3-91

S

sample programs, 5-1 to 5-6
Save command, File menu, 2-4
Save All command, File menu, 2-4
Save As command, File menu, 2-4
Save Copy As command, File menu, 2-4
Save in Text Format command, Options menu, 2-33
SavePanelState function, 4-220
scale functions. See axis scale functions.
Set Default Font command, Edit menu, 2-15
Set Target File command, Code menu, 2-23 to 2-24
Set Target File dialog box, 2-23 to 2-24
SetActiveCtrl function, 4-221
SetActiveGraphCursor function, 4-221 to 4-222
SetActivePanel function, 4-222
SetAxisRange function, 4-224 to 4-226
SetAxisScalingMode function, 4-223 to 4-224
SetCtrlAttribute function, 4-226 to 4-227
SetCtrlBitmap function, 4-227 to 4-228
SetCtrlIndex function, 4-229
SetCtrlVal function, 4-229 to 4-230
SetCursorAttribute function, 4-230 to 4-231
SetFontPopupDefaults function, 4-231 to 4-233
SetGraphCursor function, 4-233
SetGraphCursorIndex function, 4-234
SetIdleEventRate function, 3-92 to 3-93, 4-235
SetImageBits function, 4-236 to 4-238
SetInputMode function, 4-239
SetListItemImage function, 4-240
SetMenuBarAttribute function, 4-241 to 4-242
SetPanelAttribute function, 4-242
SetPanelMenuBar function, 4-243
SetPanelPos function, 4-243 to 4-244
SetPlotAttribute function, 4-244 to 4-245
SetPrintAttribute function, 4-245
SetSleepPolicy function, 4-245 to 4-246
SetSystemAttribute function, 4-226
SetSystemPopupsAttribute function, 4-247
SetTraceAttribute function, 4-247 to 4-248
SetWaitCursor function, 4-248 to 4-249
shells, building. See CodeBuilder.
Shortcut Key field, Edit Menu Bar dialog box, 2-9
shortcut keys
  constant for masking three bit fields (table), 3-28
  functions using shortcut key values as input parameters, 4-11
  producing (example), 4-157
  values for modifier keys and virtual keys, 3-26 to 3-27, 4-157 to 4-158
Show/Hide Panels command, View menu, 2-19
skeleton code
  creating with CodeBuilder, 1-4 to 1-5
  definition, 2-2
  function skeletons, 2-26
  placement in target file, 2-24, 2-26
smoothing graph updating, 3-82
Source Code Connection
  Edit Control dialog box, 2-11
  Edit Panel dialog box, 2-9
source files
  connecting code with graphical user interface, 1-3
  creating with CodeBuilder, 1-4 to 1-5, 2-2 to 2-3
speed, optimizing
  ATTR_SMOOTH_UPDATE, 3-82
  example canvas benchmark program, 5-6
  VAL_AUTO_SCALE, 3-82 to 3-83
string controls
  definition, 1-9
  illustration, 1-9
Index

operating, 1-9 to 1-10
strip chart controls
attributes
cursor styles for
  ATTR_CROSSHAIRSTYLE _STYLE
  (table), 3-78
discussion of specific attributes, 3-77
to 3-80
line styles for ATTR_LINE_STYLE
  (table), 3-79
list of attributes (table), 3-68 to 3-77
plot styles for ATTR_PLOT_STYLE
  (table), 3-80
styles for
  ATTR_CURSORPOINT_STYLE
and
  ATTR_TRACEPOINT_STYLE
  (table), 3-78 to 3-79
values for ATTR_PLOT_STYLE
  (table), 3-80
definition, 1-20
example program, 5-3, 5-5
illustration, 1-20
processing events, 3-67 to 3-68
strip chart trace functions
  ClearStripChart, 4-47 to 4-48
  function tree, 4-4 to 4-5
  GetTraceAttribute, 4-123 to 4-124
  list of functions, 3-67
  PlotStripChart, 4-176 to 4-178
  PlotStripChartPoint, 4-178 to 4-179
  SetTraceAttribute, 4-247 to 4-248
SuspendTimerCallbacks function, 4-249
swallowing events, 3-92
system attributes, 3-86 to 3-88
  list of attributes (table), 3-86
  reporting load failures, 3-87 to 3-88
  unsafe timer events, 3-86 to 3-87

T

Tab Order command, Edit menu, 2-14
to 2-15
technical support, B-1 to B-2
text box control functions
  DeleteTextBoxLine, 4-59
  function tree, 4-4
  GetNumTextBoxLines, 4-113 to 4-114
  GetTextBoxLine, 4-121 to 4-122
  GetTextBoxLineLength, 4-122 to 4-123
  InsertTextBoxLine, 4-132 to 4-133
  list of functions, 3-30 to 3-31
  ReplaceTextBoxLine, 4-215
  ResetTextBox, 4-216
text box controls
definition, 1-10
entering text, 1-10 to 1-11
example program, 5-4
illustration, 1-10
text format. See ASCII text format.
text message controls
illustration, 1-10
purpose and use, 1-10
timer control functions
  function tree, 4-6
  programming with timer controls, 3-63
to 3-65
  ResetTimer, 4-216 to 4-217
  ResumeTimerCallbacks, 4-217
  SuspendTimerCallbacks, 4-249
timer controls
  attributes, 3-64 to 3-65
definition, 1-21
  example program, 5-4
  illustration, 1-21
  operations, 3-65
timer callbacks, 3-64
timer events, unsafe, 3-86 to 3-87
toggle button controls
definition, 1-11
illustration, 1-11
operating, 1-11 to 1-12
Top Edges option
  Alignment command, 2-20
  Distribution command, 2-21
trace functions. See strip chart trace
  functions.
Track include file dependencies option, 3-1
Index

U

.uir files, 3-1
Undo command, Edit menu, 2-5
Undo Preferences section, User Interface Editor Preferences dialog box, 2-32
UnRegisterWinMsgCallback function, 4-229
unsafe timer events, 3-86 to 3-87
user interface. See graphical user interface (GUI).
user interface, creating. See graphical user interface (GUI), building.
User Interface Editor
  Arrange menu, 2-20 to 2-23
  Code menu, 2-23 to 2-30
  Create menu, 2-16
  Edit menu, 2-5 to 2-16
  File menu, 2-3 to 2-4
  Library menu, 2-30
  Options menu, 2-31 to 2-34
  Run menu, 2-30
  View menu, 2-17 to 2-19
  Window menu, 2-31
User Interface Editor Preferences dialog box
  Color Preferences section, 2-32
  Constant Name Assignment section, 2-32 to 2-33
  illustration, 2-32
  Undo Preferences section, 2-32
User Interface Editor window, 2-1 to 2-3
  Coloring tool, 2-2
  Editing tool, 2-1
  illustration, 2-1
  Labeling tool, 2-2
  Operating tool, 2-2
  popup menus, 2-2
  tool bar, 2-1 to 2-2
user interface events. See events.
User Interface Library. See also User Interface Library functions.
  color support, 4-10
  error conditions, A-1 to A-6
  font support, 4-10
  hardcopy output
    compatible printers, 4-9
    configuring devices, 4-9 to 4-10
    obtaining output, 4-9
  include file, 4-12
  plot array data types, 4-11 to 4-12
  reporting errors, 4-12
  shortcut keys, 4-11
User Interface Library functions
  function panels
    bitmap functions, 4-8
    callback functions, 4-1, 4-6 to 4-7
    canvas functions, 4-5 to 4-6
    clipboard functions, 4-8
    controls, graphs, and stripcharts, 4-1, 4-3
    function classes, 4-1 to 4-2
    LW DOS compatibility, 4-2, 4-9
    menu structures, 4-1, 4-2 to 4-3
    miscellaneous, 4-2, 4-8
    panel functions, 4-1, 4-2
    picture functions, 4-5
    pop-up panels, 4-1, 4-6
    printing, 4-2, 4-7
    rectangle and point functions, 4-7 to 4-8
    timer functions, 4-6
    user interface management, 4-2, 4-7
function reference
  AllocBitmapData, 4-12 to 4-13
  AllocImageBits, 4-14 to 4-15
  CanvasClear, 4-15 to 4-16
  CanvasDefaultPen, 4-16 to 4-17
  CanvasDimRect, 4-17 to 4-18
  CanvasDrawArc, 4-18 to 4-19
  CanvasDrawBitmap, 4-20 to 4-21
  CanvasDrawLine, 4-21 to 4-22
  CanvasDrawLineTo, 4-23
  CanvasDrawOval, 4-24 to 4-25
  CanvasDrawPoint, 4-25
  CanvasDrawPoly, 4-26 to 4-27
  CanvasDrawRect, 4-27 to 4-28
  CanvasDrawRoundedRect, 4-28 to 4-29
  CanvasDrawText, 4-30 to 4-31
  CanvasDrawTextAtPoint, 4-32 to 4-33
  CanvasEndBatchDraw, 4-34
Index

CanvasGetClipRect, 4-35
CanvasGetPenPosition, 4-35 to 4-36
CanvasGetPixel, 4-36 to 4-37
CanvasGetPixels, 4-37 to 4-39
CanvasInvertRect, 4-39
CanvasScroll, 4-40 to 4-41
CanvasSetClipRect, 4-41 to 4-42
CanvasStartBatchDraw, 4-43 to 4-44
CanvasUpdate, 4-44 to 4-45
CheckListItem, 4-45 to 4-46
ClearAxisItems, 4-46 to 4-47
ClearListCtrl, 4-47
ClearStripChart, 4-47 to 4-48
ClipboardGetBitmap, 4-48 to 4-49
ClipboardGetText, 4-49
ClipboardPutBitap, 4-49 to 4-50
ClipboardPutText, 4-50
ConfigurePrinter, 4-51
ConfirmPopup, 4-51 to 4-52
CreateMetaFont, 4-52 to 4-53
DefaultCtrl, 4-54
DefaultPanel, 4-54 to 4-55
DeleteAxisItem, 4-55 to 4-56
DeleteGraphPlot, 4-56 to 4-57
DeleteImage, 4-58
DeleteListItem, 4-58 to 4-59
DeleteTextBoxLine, 4-59
DirSelectPopup, 4-60
DiscardBitmap, 4-61
DiscardCtrl, 4-61 to 4-62
DiscardMenu, 4-62
DiscardMenuBar, 4-63
DiscardMenuItem, 4-63
DiscardPanel, 4-64
DiscardSubMenu, 4-64
DisplayImageFile, 4-65
DisplayPanel, 4-66 to 4-67
DisplayPCXFile, 4-66
DOSColorToRGB, 4-67 to 4-68
DOSCompatWindow, 4-68
DuplicateCtrl, 4-69 to 4-70
DuplicatePanel, 4-70 to 4-71
EmptyMenu, 4-71
EmptyMenuBar, 4-71 to 4-72
FakeKeystroke, 3-94, 4-72
FileSelectPopup, 4-72 to 4-74
FontSizeSelectPopup, 4-74 to 4-77
GenericMessagePopup, 4-77 to 4-79
Get3dBorderColors, 4-79 to 4-80
GetActiveCtrl, 4-80
GetActiveGraphCursor, 4-80 to 4-81
GetActivePanel, 4-81
GetAxisItem, 4-82 to 4-83
GetAxisItemLabelLength, 4-83 to 4-84
GetAxisRange, 4-84 to 4-85
GetAxisScalingMode, 4-86 to 4-87
GetBitInfo, 4-90 to 4-91
GetBitMapData, 4-87 to 4-89
GetBitMapFromFile, 4-89 to 4-90
GetCtrlAttribute, 4-91 to 4-92
GetCtlBitmap, 4-92 to 4-93
GetCtrlBoundingRect, 4-93 to 4-94
GetCtrlDisplayBitmap, 4-94 to 4-95
GetCtrlIndex, 4-95 to 4-96
GetCtrlVal, 4-96 to 4-97
GetCursorAttribute, 4-97
GetCVITaskHandle, 4-98
GetCVIWindowHandle, 4-98 to 4-99
GetGlobalMouseState, 4-99 to 4-100
GetGraphCursor, 4-100 to 4-101
GetGraphCursorIndex, 4-101
GetImageBits, 4-102 to 4-104
GetImageInfo, 4-105 to 4-106
GetIndexFromValue, 4-106 to 4-107
GetLabelFromIndex, 4-107
GetLabelLengthFromIndex, 4-108
GetListItemImage, 4-108 to 4-109
GetMenuBarAttribute, 4-109 to 4-110
GetMouseCursor, 4-110
GetNumAxisItems, 4-111
GetNumCheckedItems, 4-112
GetNumListItems, 4-112 to 4-113
GetNumTextBoxLines, 4-113 to 4-114
GetPanelAttribute, 4-114
GetPanelDisplayBitmap, 4-114 to 4-116
GetPanelMenuBar, 4-116
GetPlotAttribute, 4-116 to 4-117
GetPrintAttribute, 4-117 to 4-118
Index

GetRelativeMouseState, 4-118 to 4-119
GetScreenSize, 4-119
GetSharedMenuBarEventPanel, 4-119 to 4-120
GetSleepPolicy, 4-120
GetSystemAttribute, 4-120 to 4-121
GetSystemPopupsAttribute, 4-121
GetTextBoxLine, 4-121 to 4-122
GetTextBoxLineLength, 4-122 to 4-123
GetTextDisplaySize, 4-123
GetTraceAttribute, 4-123 to 4-124
GetUILErrorString, 4-124
GetUserEvent, 3-8 to 3-9, 3-92, 4-125
GetValueFromIndex, 4-126
GetValueLengthFromIndex, 4-126 to 4-127
GetWaitCursorState, 4-127
HidePanel, 4-128
InsertAxisItem, 4-128 to 4-129
InsertListItem, 4-130 to 4-131
InsertSeparator, 4-132
InsertTextBoxLine, 4-132 to 4-133
InstallCtrlCallback, 4-133 to 4-134
InstallMainCallback, 3-92 to 3-93, 4-135 to 4-136
InstallMenuCallback, 4-136 to 4-137
InstallMenuDimmerCallback, 4-137 to 4-138
InstallPanelCallback, 4-138 to 4-139
InstallPopup, 4-139
IsListItemChecked, 4-140
LoadMenuBar, 4-140 to 4-141
LoadMenuBarEx, 4-142 to 4-143
LoadPanel, 4-143 to 4-145
LoadPanelEx, 4-145 to 4-146
MakeColor, 4-146 to 4-147
MakePoint, 4-148
MakeRect, 4-148 to 4-149
MessagePopup, 4-149
MultiFileSelectPopup, 4-150 to 4-151
NewBitmap, 4-151 to 4-153
NewCtrl, 4-153 to 4-154
NewMenu, 4-154 to 4-155
NewMenuBar, 4-155
NewMenuItem, 4-156 to 4-158
NewPanel, 4-159 to 4-160
NewSubMenu, 4-160
PlotArc, 4-160 to 4-162
PlotBitMap, 4-162 to 4-163
PlotIntensity, 4-163 to 4-166
PlotLine, 4-166 to 4-167
PlotOval, 4-167 to 4-168
PlotPoint, 4-169
PlotPolygon, 4-170 to 4-171
PlotRectangle, 4-171 to 4-172
PlotScaledIntensity, 4-173 to 4-176
PlotStripChart, 4-176 to 4-178
PlotStripChartPoint, 4-178 to 4-179
PlotText, 4-179 to 4-180
PlotWaveform, 4-181 to 4-182
PlotX, 4-183 to 4-184
PlotXY, 4-184 to 4-185
PlotY, 4-185 to 4-187
PointEqual, 4-187
PointPinnedToRect, 4-188
PointSet, 4-188 to 4-189
PostDeferredCall, 3-94, 4-189 to 4-190
PrintCtrl, 4-190 to 4-191
PrintPanel, 4-191 to 4-193
PrintTextBuffer, 4-193 to 4-194
PrintTextFile, 4-194 to 4-195
ProcessDrawEvents, 3-93, 4-195
ProcessSystemEvents, 3-93 to 3-94, 4-195 to 4-196
PromptPopup, 4-196
QueueUserEvent, 3-94, 4-197
QuitUserInterface, 3-94, 4-197 to 4-198
RecallPanelState, 4-198 to 4-199
RectBottom, 4-199
RectCenter, 4-199 to 4-200
RectContainsRect, 4-200
RectEmpty, 4-201 to 4-202
RectEqual, 4-202
RectGrow, 4-202 to 4-203
RectIntersection, 4-203 to 4-204
RectMove, 4-204
Index

RectOffset, 4-204 to 4-205
RectRight, 4-205
RectSameSize, 4-206
RectSet, 4-206 to 4-207
RectSetBottom, 4-207
RectSetCenter, 4-207 to 4-208
RectSetFromPoints, 4-208
RectSetRight, 4-209
RectUnion, 4-209
RefreshGraph, 4-210
RegisterWinMsgCallback, 4-210 to 4-212
RemovePopup, 4-212
ReplaceAxisItem, 4-213 to 4-214
ReplaceListItem, 4-214 to 4-215
ReplaceTextBoxLine, 4-215
ResetTextBox, 4-216
ResetTimer, 4-216 to 4-217
ResumeTimerCallbacks, 4-217
RunPopupMenu, 4-218 to 4-219
RunUserInterface, 3-01, 4-219 to 4-220
SavePanelState, 4-220
SetActiveCtrl, 4-221
SetActiveGraphCursor, 4-221 to 4-222
SetActivePanel, 4-222
SetAxisRange, 4-224 to 4-226
SetAxisScalingMode, 4-223 to 4-224
SetCtrlAttribute, 4-226 to 4-227
SetCtrlBitmap, 4-227 to 4-228
SetCtrlIndex, 4-229
SetCtrlVal, 4-229 to 4-230
SetCursorAttribute, 4-230 to 4-231
SetFontPopupDefaults, 4-231 to 4-233
SetGraphCursor, 4-233
SetGraphCursorIndex, 4-234
SetIdleEventRate, 3-92 to 3-93, 4-235
SetImageBits, 4-236 to 4-238
SetInputMode, 4-239
SetListItemImage, 4-240
SetMenuBarAttribute, 4-241 to 4-242
SetMenuBarAttribute, 4-241 to 4-242
SetPanelAttribute, 4-242
SetPanelMenuBar, 4-243
SetPanelPos, 4-243 to 4-244
SetPlotAttribute, 4-244 to 4-245
SetPrintAttribute, 4-245
SetSleepPolicy, 4-245 to 4-246
SetSystemAttribute, 4-246
SetSystemPopupsAttribute, 4-247
SetTraceAttribute, 4-247 to 4-248
SetWaitCursor, 4-248 to 4-249
SuspendTimerCallbacks, 4-249
UnRegisterWinMsgCallback, 4-229
ValidatePanel, 4-250
WaveformGraphPopup, 4-251
XGraphPopup, 4-252
XYGraphPopup, 4-252 to 4-253
YGraphPopup, 4-253
user interface management functions
FakeKeystroke, 3-94, 4-72
function tree, 4-7
GetSleepPolicy, 4-120
GetUserEvent, 3-92, 4-125
ProcessDrawEvents, 3-93, 4-195
ProcessSystemEvents, 3-93 to 3-94, 4-195 to 4-196
QueueUserEvent, 3-94, 4-197
QuitUserInterface, 3-94, 4-197 to 4-198
RunUserInterface, 3-01, 4-197 to 4-198
SetIdleEventRate, 3-92 to 3-93, 4-235
SetInputMode, 4-239
SetSleepPolicy, 4-245 to 4-246
user interface objects, finding, 2-17 to 2-18
user interface resource (.uir) files, 3-1

V

VAL_AUTO_SCALE, 3-82 to 3-83
validate control mode for commit events
definition, 1-4
requirements, 1-4
ValidatePanel function, 4-250
Vertical Centers option
Alignment command, 2-21
Distribution command, 2-21
Vertical Compress option, Distribution command, 2-22
Vertical Gap option, Distribution command, 2-21
View button, Edit Menu Bar dialog box, 2-9
View command, Code menu, 2-28 to 2-29
View menu, User Interface Editor
  Bring Panel to Front command, 2-19
  Find UIR Objects command, 2-17 to 2-18
  illustration, 2-17
  Next Panel command, 2-19
  Preview User Interface Header File command, 2-19
  Previous Panel command, 2-19
  Show/Hide Panels command, 2-19
virtual keys
  representation in source code, 3-26
  values for shortcut keys, 3-26 to 3-27, 4-157 to 4-158

Y

Y axes
  autoscaling Y-axis on strip chart, example, 5-5
  example programs for left and right Y axes, 5-5
  working with two Y axes, 3-81
YGraphPopup function, 4-253

Z

zooming on graphs
  definition, 1-19
  enabling zooming, 1-19

W

WaveformGraphPopup function, 4-251
Window menu, User Interface Editor, 2-31
windows interrupt support functions
  function tree, 4-7
  GetCVITaskHandle, 4-98
  GetCVIWindowHandle, 4-98 to 4-99
  RegisterWinMsgCallback, 4-210 to 4-212
  UnRegisterWinMsgCallback, 4-229
Windows metafiles, 3-62 to 3-63
Wrap option, Find UIR Objects dialog box, 2-18

X

XGraphPopup function, 4-252
XYGraphPopup function, 4-252 to 4-253